Programming by Demonstration of Robot Manipulators

To my family

Örebro Studies in Technology 34

Alexander Skoglund

Programming by Demonstration of Robot Manipulators

© Alexander Skoglund, 2009

Title: Programming by Demonstration of Robot Manipulators

Publisher: Örebro University 2009
www.publications.oru.se

Editor: Jesper Johanson
jesper.johanson@oru.se

Printer: Intellecta Infolog, V Frölunda 05/2009

ISSN 1650-8580
ISBN 978-91-7668-669-0

Abstract

If a non-expert wants to program a robot manipulator he needs a natural interface that does not require rigorous robot programming skills. Programming-by-demonstration (PbD) is an approach which enables the user to program a robot by simply showing the robot how to perform a desired task. In this approach, the robot recognizes what task it should perform and learn how to perform it by imitating the teacher.

One fundamental problem in imitation learning arises from the fact that embodied agents often have different morphologies. Thus, a direct skill transfer from human to a robot is not possible in the general case. Therefore, a systematic approach to PbD is needed, which takes the capabilities of the *robot* into account–regarding both perception and body structure. In addition, the robot should be able to learn from experience and improve over time. This raises the question of how to determine the demonstrator's goal or intentions. It is shown that this is possible–to some degree–to infer from multiple demonstrations.

This thesis address the problem of generation of a reach-to-grasp motion that produces the same results as a human demonstration. It is also of interest to learn what parts of a demonstration provide important information about the task.

The major contribution is the investigation of a *next-state-planner* using a *fuzzy time-modeling approach* to reproduce a human demonstration on a robot. It is shown that the proposed planner can generate executable robot trajectories based on a generalization of multiple human demonstrations. The notion of *hand-states* is used as a common motion language between the human and the robot. It allows the robot to interpret the human motions as its own, and it also synchronizes reaching with grasping. Other contributions include the model-free learning of human to robot mapping, and how an imitation metric can be used for reinforcement learning of new robot skills.

The experimental part of this thesis presents the implementation of PbD of pick-and-place-tasks on different robotic hands/grippers. The different platforms consist of manipulators and motion capturing devices.

Keywords: programming-by-demonstration, imitation learning, hand-state, next-state-planner, fuzzy time-modeling approach.

Acknowledgement

First, I would like to thank my supervisors, for their help, our discussions and most importantly their scientific guidance. My supervisor Rainer Palm, for the valuable inspiring conversations, teaching me about robotics, and introducing me to important methods. My assisting supervisor, Boyko Iliev, has been a great source of knowledge and cooperation as well as inspiration. I am very thankful to my supervisors for proofreading and commenting on several versions of each chapter, and correcting embarrassing mistakes.

I would also like to thank Johan Tegin at the Royal Institute of Technology, Stockholm, for both the assistance in paper writing, valuable feedback, general discussions, and for providing access to the KTHand which was a most useful tool. I thank Jacopo Aleotti at Parma University for the collaboration we had during his stay in Örebro in 2004. Dimitar Dimitrov should be acknowledged for his outstanding knowledge in robotic simulation and control, and for always being helpful and sharing his time. Without Krzysztof Charusta, my final experiments would not have been possible–thank you! I'm also happy to have worked with Tom Duckett, Achim Lilienthal, Bourhane Kadmiry and Ivan Kalaykov during my years as a Ph.D. student.

Our research engineers Per Sporrong and Bo-Lennert Silfverdahl should be acknowledged for their help in the lab with our robots and motion capturing systems. All the Ph.D. students at AASS should also be acknowledged for the great social environment they create, both during–and often after–work.

Finally, I would like to thank my wife, Johanna, for all her love, for her support, and for being my best friend, although I am not alway present. And my daughter Juni for her love, laughs and warm welcome; although she thinks I work with "sopor"[1] because I bring them out almost every morning.

Örebro, March 25:th, 2009
Alexander Skoglund

[1] Swedish for garbage.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Today when the robotics community is finding more and more applications for robots it would be beneficial to make these machines easier to use than it currently is. Since it would be hard to program such a general robot to anticipate every possible situation or task it may encounter, the user must be able to instruct the robot in a natural and intuitive way, instead of to program it like a computer programmer does. One such way is to instruct the robot what to do using our own body language and knowledge of the task. For humans it is easy to imitate a movement or a task shown, so one could assume that imitation would be an easy task for state-of-the-art robots (nowadays), considering computing power and sensors available today. However, a robot that is able to learn new skills that we as humans consider to be simple has been very hard to accomplish. It turns out to be very hard for a robot to acquire even elementary skills by imitation, to quote Schaal [1999]:

> "Interestingly, the apparently simple idea of imitation opened a Pandora's box of important computational questions in perceptual motor control."

The ability to imitate is very well developed in humans and some primates but rarely found in other animals. And given a second thought–think of a pet–it is not straight forward to just show your dog or cat how to fetch the newspaper. So it should be no surprise that it has proven very hard to design a robot with the same imitating capabilities as humans. But, it is an appealing thought to have a robot to be able to learn from and improve upon human demonstration. Therefore, there is a growing interest in robots that can be programmed just by observing a task demonstrated to it. This is a scientific challenge for robotic scientists. It would save much time otherwise spent on programming the robot.

A topic related to imitation is intention recognition. To understand the user's intention is a very difficult task and will not be addressed directly in this thesis. Most of this thesis focus on *how* to perform learning from demonstration.

1.1 Robot Programming

Biggs and MacDonald [2003] classified programming of industrial robots in two categories: *manual-* or *automatic programming*. In manual programming a text- or graphical interface control the robot. "Automatic programming" means to automatically create the program that controls the motion executed by the robot, thus, the user affects the robot's behavior instead of the program. The most common way to program an industrial robot is to guide the robot manually using a teach pendant (an advanced joystick) or a similar device. It is also possible to move the manipulator manually by hand to record joint positions and trajectories. This approach belongs to manual programming since the programmer typically needs to manually (e.g., using a text editor) modify the final trajectory, which allows the programmer to have extensive control over the task.

The concept to simplify robot programming is the so-called *Programming--by-Demonstration*-paradigm, PbD, which enables the robot to imitate a task shown by a teacher. The demonstration can be performed in several ways, by manually guiding the robot, by teleoperating it with a remote control or by a demonstrator performing the task without any interaction with the robot by a so-called motion capturing system, which records human motions. Kuniyoshi et al. [1994] was one of the earliest researchers considering the imitation programming, where the user pursue the task in a "natural" way, not by guiding a robot.

In a study by Asada et al. [2001], they approached the subject PbD in a cognition context, where the robot should develop cognitive skills by imitating humans.

1.2 Motivation

The motivation for the work described in this thesis is to develop new PbD methods and improve existing ones in order to facilitate easy programming of robotic manipulators. In industry, manipulators are part of automation lines used mainly by companies that make high volume products or products that require high repeatability in the assembling task. These automation solutions, designed for high volume manufacturers, are typically both expensive and complicated. Depending of the task the robot has to perform; the programming process can be both difficult and time-consuming. Small and medium sized enterprises (SME) are unlikely to invest in an expensive robot and reprogram it when their products change, unless the transition from assembling or handling of one product to the other requires much less effort than performing the work manually.

PbD provides simple programming of industrial robots, thus removes one impediment for SMEs with small production series to automating their produc-

tion. This is also important for large companies because even they may have an internal structure of several SMEs.

When a robot acquires new knowledge using a pure machine learning strategy, it most probably will be cumbersome to learn a task without prior knowledge, if it is possible at all. Indeed, Schaal [1997] showed that PbD speeds up certain types of machine learning problems.

Another reason for using learning from a demonstration by imitation is if the expert's knowledge for doing some task is not straight forward to put into rules, or the rules get too complicated to deal with. In other words, the teacher cannot explicitly tell the learner what it should learn.

1.3 Key Questions

There are five main problems involved in the general formulation of skill transfer by imitation Nehaniv and Dautenhahn [2002]. These are *who*, *when*, *what*, *how to map* and *how to evaluate*.

Who to imitate? This problem is about defining who is the teacher. In this thesis, the teacher is defined by who is wearing the motion capturing device.

When to imitate? This problem concerns when it is appropriate to imitate. In this thesis, we address this question on a short time scale by judging what parts of the trajectory that is important for a specific task.

What and how to map? These two questions involve how to create a mapping from human body motions to robot motions, and what impact such actions have on the world. The morphological differences–the fact that the demonstrator and the robot are not the same person–raises a correspondence problem, which is a major challenge [Nehaniv and Dautenhahn, 2002]. These questions are of main interest in imitation research in general, and addresse the actual execution of a skill or a task–the main scope of this thesis–and the purpose of a task.

How to evaluate? Probably the hardest question to answer, since this means to find a metric by which the robot can evaluate the performance of its own actions. In this thesis we will use a predetermined metric, partially derived from demonstration, for skill evaluation.

To record a trajectory with a motion capturing device and then replay it to the manipulator might seem as an easy way to obtain the demonstrated motion. Compare this to recording a sound like music or a talk. The sound can be replayed and exactly mimicked the original recording, but to claim that the device that replays the recording can play music, talks or have a philosophical monolog is—to say the least–far fetched. Analogously, to replay a recorded motion on a robot does not account for generalization capability, morphological differences, noise or any other feature that machine leaning can provide the

robot with; it *only* replays the motion. Furthermore, neither the motion capturing nor the robot controller is perfect, thus, an exact reproduction becomes impossible [Atkeson and Schaal, 1997].

The difference in the location of the human demonstrator and the robot might force the robot into unreachable parts of the workspace or unreachable arm configurations even if the demonstration is perfectly feasible from human viewpoint. Thus, it is not possible to transfer the human task demonstration directly as a desired motion trajectory for the robot. Instead, it serves as a source of information about the target object as well as the basic properties of the arm and hand motions during reach and grasp.

Other challenges stem from the robot's lack of a *theory of mind*, thus, the robot cannot know the intention of the user. However, from a series of demonstrations performed with variation the robot can infer what part(s) of a demonstration *seem* to be important and what is of less importance. To give reaching motions a meaning grasping is discussed in this thesis in the context of generating a reaching motion suitable for grasping. Reaching motions are a complement to grasping as they provide both hand-trajectory and a proper positioning for the end-effector to perform a grasp.

Many articles discussed in this thesis concern reaching and pointing but not grasping, and generally only investigates reaching in two dimensions. In this context this thesis provides an extension since we consider all six dimensions to position and orient the end-effector to execute a grasp.

1.4 Objectives

Our long term objective is to develop an architecture that makes instruction of a robotic manipulators easy. It should be possible to teach the robot new basic skills. These basic skills should then be used for the composition of novel tasks, more complex than the basic skills. The initial skills, acquired from the teacher, should also self-improve during operation and possibly become better than the initial skill.

The short-term objective was to investigate how to use learning to control a manipulator, using a motion capturing system. We consider simple operations such as pick-and-place first after the robot have learned some "basic" skills like move-to-point (reaching). The aim of the work in this thesis is to investigate learning of reaching skills and basic manipulation tasks from human demonstrations. One important benefit derived from the PbD method is the humanlike appearance of the motion which implicitly also augments safety since the motion is predictable to humans (in contrast to, e.g., time-optimal motions).

The more specific objectives of this thesis include:

- To address the correspondence problem resulting from morphological differences between the human and the robot.

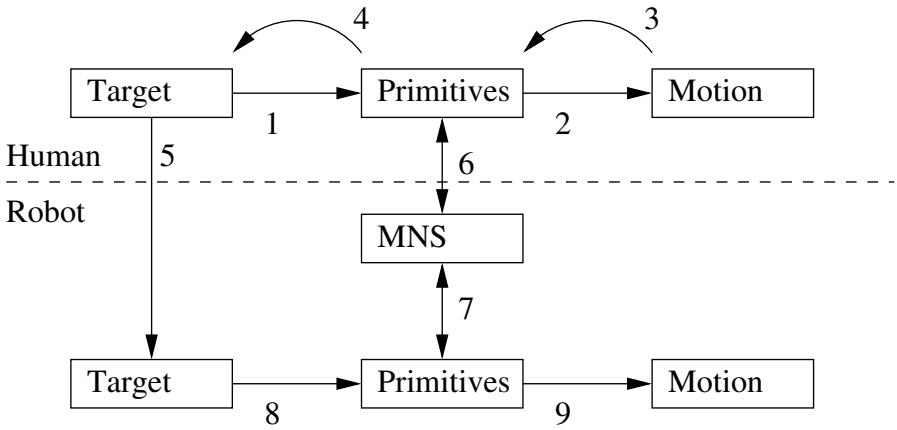


Figure 1.1: *The imitation process in our approach.*

- Enable skill transfer from humans to robots, despite morphological differences.
- Preserve the key characteristics of human motion during the transfer into robot motions.
- Find a skill description which is general enough to allow the robot to learn from both human demonstrations and own experience in the same modeling framework.

1.5 Our Approach to PbD

To illustrate the *applicability* of our approach the platform requires only hardware available of-the-shelf: a general-purpose motion capturing system and an industrial robot. Industrial manipulators are mature products and thus very reliable in operation. We use commercially available motion capturing systems for the demonstration of motions from the teacher. We describe our approach as an imitation process. This process, illustrated in figure 1.1, is the framework that has evolved during our work. Each transition in this process is enumerated, here are the explanations:

1. The human operator intends to do some task involving object manipulation, thus having a *target* object within the task, such as "move object A, from point B to C, in a way described by the trajectory D". From the start this target is hidden from the robot–and other humans–since it is not observable. When the human performs the task a number of *motion primitives* are activated.

2. A motion primitive is a small unit of behavior that, for example, can encode a part of a trajectory or a specific motion type such as "Move down". Several motion primitives are combined to perform a task, a skill or just a motion.
3. The motion that the human performs *is* observable. A set of primitives–underlying the human motion–can be used to *encode* the trajectory.
4. Some aspects of the target also become observable, such as which object should be moved and, to some extend, in which way.
5. When the target can be observed, the robot has access to this knowledge.
6. In addition, the *mirror neuron system*, MNS, provides a link between the human primitives and the robot primitives. The human side of the MNS system has primitives encoded to fit the human motion.
7. In a similar way, primitives are encoded to fit the robot's morphology. If the observed motion is novel to the robot new primitives are created. i.e., a new primitive has been *learned*.
8. The target and the MNS together activate the primitives.
9. The final motion is executed by the robot's own primitives.

1.6 Contributions

One of the main contributions of this thesis is the creation of a PbD architecture on the basis of the following methods and approaches:

- We evaluate learning of a sensor mapping from a demonstration to a robot manipulator without any knowledge of the demonstrator's arm configuration.
- We introduce of *fuzzy time-modeling* for encoding demonstrated trajectories.
- In addition, we also introduce of distance dependent variability.
- We introduce an application of a *next-state-planner* based on the fuzzy-clustering principle, which plans actions "on-the-fly" instead of complete trajectories.
- We use the notation of *hand-states* applied to PbD, for a coherent description of prehension which synchronizes reaching and grasping.
- We show how a demonstration can speed up learning of a reaching task, using reinforcement learning.
- A skill metric is introduced to adapt a skill to the morphology of the robot, after the initial learning.

1.7 Terminology

In the robot imitation literature terms often have different meanings in different but similar contexts, especially when dealing with terms like "primitive", "skill" and "task". To avoid confusions of terminology this thesis will use the terminology listed below.

Primitive A *primitive* is a small unit of behavior. In this thesis a primitive refers to the cluster centers that encode a motion, which is the smallest piece of motion considered.

Skill A *skill* is encoded by a sequence of primitives, e.g., reaching for a cylindrical object.

Task A *task* is a sequence of skills such as pick-and-place.

1.8 Publications

The work in this thesis has been presented in a number of publications:

- Alexander Skoglund, Johan Tegin, Boyko Iliev and Rainer Palm *Programming-by-Demonstration of Reach to Grasp Tasks in Hand-State Space* Accepted for publication at 2009 International Conference on Advanced Robotics (ICAR 2009), Munich, Germany, June 22-26.
- Johan Tegin, Boyko Iliev, Alexander Skoglund, Danica Kragic and Jan Wikander *Real Life Grasping using an Under-actuated Robot Hand–Simulation and Experiments* Accepted for publication at 2009 International Conference on Advanced Robotics (ICAR 2009), Munich, Germany, June 22-26.
- Alexander Skoglund, Boyko Iliev and Rainer Palm. *A Hand State Approach to Imitation with a Next-State-Planner for Industrial Manipulators* Presented at 2008 International Conference on Cognitive Systems, Karlsruhe, Germany, April 2-4. *Will be published by Springer in the first edition of the Springer series "Cognitive Systems Monographs".*
- Alexander Skoglund and Boyko Iliev. *Programming By Demonstrating Robots Task Primitives* SERVO Magazine, December 2007. Not peer-reviewed.
- Alexander Skoglund, Boyko Iliev, Bourhane Kadmiry and Rainer Palm. *Programming by Demonstration of Pick-and-Place Tasks for Industrial Manipulators using Task Primitives* Presented at 2007 IEEE International Symposium on Computational Intelligence Robotics and Automation, Jacksonville, Florida, June 20-23.

- Alexander Skoglund, Tom Duckett, Boyko Iliev, Achim Lilienthal and Rainer Palm. 2006 *Teaching by Demonstration of Robotic Manipulators in Non-Stationary Environments* Presented at 2006 IEEE International Conference on Robotics and Automation Orlando, Florida, US, May 15-19.

- Alexander Skoglund, Rainer Palm and Tom Duckett. 2005 *Towards a Supervised Dyna-Q Application on a Robotic Manipulator* Proc. SAIS-SSLS 2005, 3rd Joint Workshop of the Swedish AI and Learning Systems Societies, Västerås, Sweden, April 12-14.

- Jacopo Aleotti, Alexander Skoglund and Tom Duckett. 2004. *Teaching Position Control of a Robot Arm by Demonstration with a Wearable Input Device* Proceeding IMG04, International Conference on Intelligent Manipulation and Grasping, Genoa, Italy, July 1-2.

1.9 Outline of the Thesis

The rest of this thesis is organized as follows:

Chapter 2 Methods in Imitation Learning We start with an overview of the field of imitation learning and its application to PbD. This chapter reviews methods related to the approach in this thesis.

Chapter 3 Supervised Learning in Programming-by-Demonstration This chapter describes two approaches to PbD based on supervised learning. The first is a prototype of a PbD system for position teaching of a robot manipulator. This method does not require analytical modeling of neither the human arm nor robot, and can be customized for different users and robots. A second approach is also presented where a known task type is demonstrated and interpreted using a set of skills. Skills are basic actions of the robot/gripper, which can be executed in a sequence to form a complete a task. For modeling and generation of the demonstrated trajectory a new method called fuzzy time-modeling is used to encode motions, resulting in smooth and accurate motion models.

Chapter 4 Trajectory Generation in Hand-State Space Here, we present an approach to reproduce human demonstrations in a reach-to-grasp context. The demonstration is represented in hand-state space, which is an object centered coordinate system. We control the way in which the robot approaches the object by using the distance to the target object as a scheduling variable. We formulate the controller that we deploy to execute the motion as a next-state-planner. The planner produces an action from the current state instead of planning the whole trajectory in advance which can be error prone in non-static environments. The results have a direct application in PbD.

Chapter 5 Reinforcement Learning for Reaching Motions In this chapter we investigate how demonstrations can be used to speed up learning of a reaching

task, in a reinforcement learning framework. Hence, the usually slow reinforcement learning is speed up by using the demonstration to guide the learning agent. Furthermore, a developmental approach is presented which combines the fuzzy modeling based next-state-planner from chapter 4 with reinforcement learning. From an imitation metric, based on the hand-state error and the success-rate in a reach to grasp task, the robot develops new skills from executing a model of the observed motion.

Chapter 6 Conclusions This chapter concludes this thesis by a summary and discussion. We also propose directions for future research.

Chapter 2
Imitation Learning

Imitation learning refers to the process of learning a task, skill or action by observing someone else and reproduce it. This learning process should not be confused with the concept of *machine learning* since the former uses the term "learning" in a general sense while the latter is a collection name for algorithms used to extract knowledge from data. Different machine learning techniques can be (and are!) used as tools for imitation learning, such as instance-based learning and reinforcement learning both applied by e.g., Atkeson and Schaal [1997], and learning a set of rules [Ekvall and Kragic, 2006].

Programming-by-Demonstration (PbD) provides a natural way to program robots by showing the desired task. PbD is an application of imitation learning meaning that the robot must imitate the demonstrated motion by first interpreting the demonstration and then reproduce it with its own action repertoire. There are three main benefits offered by PbD. Firstly, as robots become more common for a specialized task, they are not yet suited as general purpose machines for everyday use. One of the main reasons is that they are difficult to program. They require expert users to do this job while it should be as easy as asking a colleague to help you. Secondly, imitation learning provides a reduced search space for machine learning algorithms, thus limiting complexity and reducing time to learn a task. Finally, models of perception coupling and actions, which are at the core of imitation learning could help us to understand the underlying concepts of imitation in biological systems.

2.1 Programming by Demonstration Systems, An Overview

In a general setting a PbD environment consists of a robot manipulator and a motion capturing system. Methods and equipment are built on different principles. For example, the manipulator can be guided through kinesthetic demonstration which means that the teacher moves the manipulator manually while the robot observes the joint angles of the arm [Calinon et al., 2007]. A similar

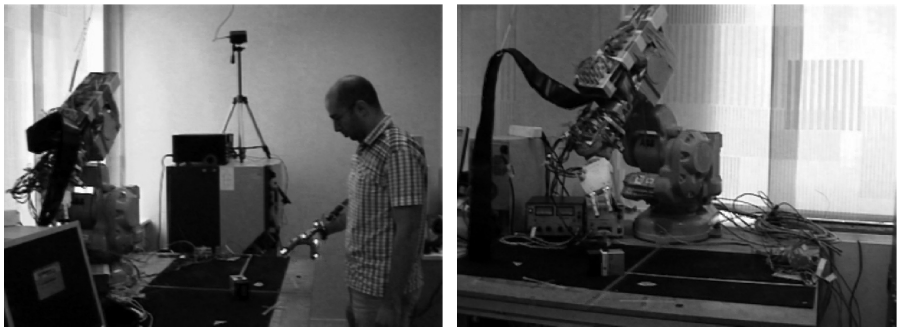


Figure 2.1: *Industrial manipulator programmed using a demonstration.*

method is to guide the manipulator using a teach pendant or a joystick, which is a common method in mobile robotics [Li and Duckett, 2005]. Another method for data recording is to capture the motions of a human, while performing a demonstration. However, this comes at the cost that the robot might not be able to imitate the teacher because of differences in morphology and configuration. Therefore, the teacher must be aware of this in order to produce a meaningful demonstration. Going one step further is to only rely on robot vision to acquire the demonstration [1] which typically suffers from occlusions and inaccuracy. In the above data acquisition process there are two different ways to do imitation. One way is to learn directly by imitating, which requires the imitation capability to be there from the start. The other way is to observe the whole demonstration first and make the imitation afterwards. Typically, the type of task determines which approach is more applicable.

To shed some light on the roots of the PbD paradigm a brief history is provided. These papers represent a snapshot of state-of-the-art for their respective time:

1984 Dufay and Latombe [1984] proposed a teaching approach based on symbolic AI, where a symbolic planner generated a sequence of instructions. They did not perform data capturing of the human motion, the joint positions of the robot and a force sensing wrist were instead used to capture sensor data.

1994 Kuniyoshi et al. [1994] outlined a system with motion capturing sensors that records the human in a natural posture. The robot reproduced the recorded task.

2004 Tani et al. [2004] showed a strategy with phases for learning, action and recognition, implemented on a humanoid robot for behavior control taught with kinesthetic demonstrations.

[1] Several motion capturing systems are based on vision, however, these systems require the user to wear some special makers or colors. A "true vision system" resembles biological vision systems that do not require engineered environments or cameras distributed in the environment.

Today's research in imitation learning often address one or several of what has been identified as key questions in imitation learning. These are five central issues in imitation, elegantly identified by Dautenhahn and Nehaniv [2002b], who dubbed them the "big five". These are *who*, *when*, *what*, *how to map* to imitate, and *how to evaluate* an imitation.

***Who* to imitate** This problem concerns who is a good expert for building a model and is linked with the general perception problem that robotics generally deal with. Most studies in robotic imitation avoid or circumvent questions regarding "who to imitate" and if "the learner can become the teacher". By the nature of most motion capturing systems only one person's motions is recorded and per definition used for teaching. In doing so, both the perception- and *who* problems are avoided. This is the case for the experiments presented in this thesis. Given that there are off-the-shelf products (such as BioStage from OrganicMotion[2]), offering markerfree motion capturing, the author believes that the computer vision community will provide a solution to the perception problem in *restricted environment and usage* very soon.

***When* to imitate** This is about knowing how to distinguish when it is appropriate to imitate and when not. In the context of interaction, the robot should also be able to distinguish a demonstration from irrelevant motions. Also, the robot should be able to know when to imitate on a short time scale, e.g., the beginning of a motion or the end, or both.

***What* to imitate** What was the purpose, or the goal, in performing an action? This involves the problem of recognizing the intention of the action [Jansen and Belpaeme, 2006]. In the context of reaching and grasping, it is important to know what parts of a demonstration are relevant in order to grasp an object. If an agent is skilled in interacting with the world, imitation can be performed on this level.

***How* to map** a demonstration into an imitation when the teacher and learner have different morphologies, means that the correspondence between the two has to be established. Consider, for example, a standard industrial manipulator configuration, called "elbow up", which is quite different from the human arm with an "elbow down" configuration. If a direct mapping of joint coordinates from the teacher to the imitator is made, there will be a mismatch in joint coordinates since there is a difference in morphology. Hence, a *correspondence problem*[3] occurs in the case of a mismatch between teacher and imitator.

To *evaluate* an imitation The ability to self-evaluate an imitation would enable the robot to self-improve its skills obtained from demonstration. Several

[2] www.organicmotion.com

[3] In the literature on imitation this is known as "the correspondence problem".

researchers have suggested different metrics to evaluate the demonstration and clearly the metric is context-depended. While studies have addressed this issue in a task specific context, there is to the knowledge of the author, no study yet to approach this question in a holistic manner.

2.2 Biologically Inspired Methods

Biological systems often serves as inspiration for approaches to robotic imitation learning. Biologically inspired imitation systems are typically classified into two groups: conceptual and connectionist models. The latter look at the biological system at a neuron level–the actual neural mechanisms–and build systems that are biologically plausible both in function and structure, thus usually uses artificial neural networks. The former approach, conceptual modeling, looks at the *function* of a biological system and builds artificial systems that mimic the function of the biological counterpart. These models are typically based on findings from, for example, neuroimagining studies, where an area is shown to be associated with some phenomena. In the conceptual approach any artificial mechanism, for example, support vector machines or fuzzy modeling, could be used without biological motivation. However, on a *system level* it has equivalent properties as its biological counterpart. The approach that we will describe later in this thesis, specifically chapter 4, is inspired from the mirror neuron system, and follows that system model from a conceptual approach.

In nature, imitation occurs in many different ways. In evolutionary imitation, for example, a harmless insect can evolve to look like some other poisonous or dangerous insect. On the other side of the time spectrum is instantaneous behavioral imitation that parents are familiar with: "Don't do like I do. Do what I say!"–a strategy that is far from perfect. When a match occurs between an observed motion and a motor action already existing in the observer's motor repertoire it is called *response facilitation*. According to Byrne [2003], this is different from *true imitation* where a novel action is observed and reproduced. Many robotic approaches to imitation do classification/recognition of known skills, and should consequently be called "response facilitation systems" instead of "imitation systems". Response facilitation systems involve the correspondence problem in which the mapping from the observed behavior has to fit some existing motor action. However, from a task point of view a new task can be performed by following a demonstration where only the basic components of the task (the skills) are known beforehand.

A *mirror neuron* is a type of neuron that fires both when the subject observes a demonstration *and* when it performs the same action as the observed [Rizzolatti et al., 1996]. Hence, the mirror neuron system provides a link from visual input to motor actions in the context of reaching and grasping motions. These neurons which are scattered over different areas of the brain were first discovered in monkeys and later also in the human brain. It is not yet clear if these neurons are what makes imitation possible or if they just participate

in the motor reproduction of movements [Rizzolatti, 2005, Brass and Heyes, 2005].

Dautenhahn and Nehaniv [2002a] hypothesize that mirror neurons are nature's own solution to the correspondence problem, in that they are matching mechanisms for mapping observations (visual input) to execution (motor output). Zentall [2007] criticizes the mirror neuron hypothesis for being unclear about the underlying mechanisms that makes these neurons fire.

Oztop and Arbib [2002] propose a functional model of the mechanisms behind primate grasping. Furthermore, they introduce the Mirror Neuron System (MNS) model to explain the function of the MNS located in Brodmann's brain area F5, which is related to grasping. The model uses the notion of affordances, defined as *object features relevant to grasping*. This suggests that the mirror neuron system in F5 uses an internal representation of the action, which is independent of who executed the action. In accordance to that, they introduced the term hand-state defiend as a vector whose components represent the movement of the wrist relative to the location of the object and of the hand shape relative to object's affordances. Consequently, grasp motions are modeled with hand-object relational trajectories as internal representations of actions.

2.2.1 Architectures for Human Motions

Given a task such as picking up a cup and drinking, the human's central nervous system (CNS) can execute the task in infinitely many ways. However, despite the number of possibilities human motions are highly stereotypical, both across one individual performing the task several times and across a number of individuals performing the same task. This has lead researchers to search for a model that describes "human motion".

Movement primitives have been proposed as small units of behavior on a higher level than motor commands. By combining these primitives a full motor repertoire can be executed, which is an attractive thought from a computational point of view. Evidence from neuroscience shows that that movement primitives actually exist in nature [Bizzi et al., 1995]. They demonstrated that only a small number of primitives are enough to encode a frog's motor scheme, where a small number of force fields represents the whole workspace of the frog's leg. In an artificial imitation framework, a motion primitive could be symbolic description like "Move down" [Aleotti et al., 2004]. They can also be described as parametric models of policies that in combination are capable of achieving a complete movement behavior [Schaal, 1999]. In robotics it is desirable to have a limited set of primitives that can be combined to arbitrary tasks, either sequentially or in parallel. On a *symbolic level* (see section 2.4) these primitives can either be predefined, discussed in chapter 3, or learned, discussed in chapter 4.

Bizzi et al. [1995] and Mussa-Ivaldi and Bizzi [2000] put forward the *equilibrium point hypothesis* where modules in the spinal cord are viewed as computational primitives for motion. From experimental studies on frogs where electrical stimulation is applied to the premotor circuits in the spinal cord, Bizzi et al. [1991] showed that these circuits are organized in a set of discrete modules. Furthermore, Mussa-Ivaldi et al. [1994] showed that the principle of vectorial summation can be applied to leg moments in frogs and rats, when two of the modules are simultaneously stimulated. According to this hypothesis, a motion trajectory is produced by temporal series of equilibrium points, called "virtual trajectory".

Wolpert and Ghahramani [2000] suggest the notion of *internal models*, or *inverse internal models*, also called controllers, or behaviors. An internal model is the central nervous system model of the mapping from a desired consequence (sensory information) to an action (motor command). The opposite mapping is the *forward model* or *predictor* predicting the consequence of an action. Wolpert et al. [1998] hypothesized that the internal models are located in the cerebellum and that an inverse model of the dynamics is learned when an object is manipulated. Furthermore, Wolpert and Kawato [1998] proposed a modular approach to describe human motor learning and control, called MOSAIC (MOdule Selection And Identification for Control).

Samejima et al. [2006] used MOSAIC to perform a swing up task for a simulated two link robot with one actuator. Four modules, each with a controller and a predictor, were trained by observing the swing up task using a *responsibility signal* which decides how much each module contribute to the final control signal. To train the state prediction and control reinforcement learning was used.

2.2.2 Characteristics of Human Motions

Often, robot trajectories are time optimal or planned to follow some joint space restrictions, which may lead to trajectories that are unlike human motions. However, in many cases it is desirable to make robots moving humanlike for two reasons. Firstly, if robots and humans shall interact, humans must be able to predict the robot's motions and thereby "intention". This is a matter of safety[4] and comfort for the human operator. Secondly, to follow a demonstrated motion it is desirable that the motions produced by the robot controller have humanlike characteristics to conform with the original motion.

A number of specific characteristics which are all associated with human motions can be modeled and applied to robots accordingly.

Fitts [1954] formulated a law describing the tradeoff between speed and accuracy of the human hand motion in mathematical terms. It states that the duration of a reaching motion to a target is related to the amplitude of the

[4]In todays factories robots must be placed behind fences.

movement and the size of the target (the precision of the motion). For a reach-to-target motion the amplitude is equal to the distance to the target at the start of the motion. This relation is formulated by Fitts' law as:

$$MT = a + b \log_2 \left(\frac{2A}{W} \right) \tag{2.1}$$

where MT is the duration of the motion, A is the amplitude of the motion, equal to distance to the target at the start of the motion, W is the width of the object and a and b are coefficients. By applying Fitts' law, a robot can preserve the equivalent duration of movement as from the observed demonstration and compute an estimated movement duration.

Flash and Hogan [1985] investigated voluntary human arm movements. The subject used a mechanical manipulandum (a passive robot) to record planar, multi-joint reaching motions. They described human motions mathematically as *minimum jerk* motions, which is the derivative of the acceleration. By assigning a cost to each function the one with the lowest cost is selected. For a one dimensional problem that is to minimize:

$$C(x(t)) = \frac{1}{t_f} \int_{t_0}^{t_f} \dddot{x}^2 \, dt \tag{2.2}$$

where C is the cost, $x(t)$ is the end-effector location as a function of time, $\dddot{x}$ is the jerk and t_0 and t_f are the times of start and reaching the final goal respectively. This can be described as a *control policy*:

$$\begin{aligned} \Delta\ddot{x}_d(t) &= \left(-\frac{60}{D^3}(x_t - \hat{x}(t)) - \frac{36}{D^2}\dot{x}_d(t) - \frac{9}{D}\ddot{x}_d(t) \right) \\ \Delta\dot{x}_d(t) &= \ddot{x}_d(t) \\ \Delta x_d(t) &= \dot{x}_d(t) \end{aligned} \tag{2.3}$$

where $D = t_f - t$ the remaining time to reach the goal, x_d is the desired location at time t and $\hat{x}$ is the current estimate of the end-effector location, adopted from Shadmehr and Wise [2005]. Equation 2.3 computes the changes in acceleration, velocity, and position as a function of its current position and the time being left to reaching the goal, that is a *next-state-planner*, which will be described in section 2.7.

By recording arc shaped drawing motions in a 2-dimensional plane (both constrained and unconstrained), Lacquaniti et al. [1983] showed that the tangential velocity of the human hand is dependent on the curvature of its movement. More specifically, the velocity $V(t)$ of the motion is dependent of the radius $R(t)$ of the curvature at each segment where the relation can be described by the the *two-third power law*:

$$V(t) = kR(t)^{1-\beta} \tag{2.4}$$

where k is a gain factor and $\beta \approx 2/3$ is the constant giving this relationship its name.

When a human grasps an object and moves it to a final location, she performs the movement so that she reaches the orientation of the end-state in a most comfortable way. Rosenbaum et al. [1996] called this the *end-state comfort*. However, the end-state comfort does not apply when a subject first grasps an object and transport it to some other location, and then grasp it again and transport it back. Instead, the subject remembers the end-effectors posture at the end-state and applies this posture instead of what would be the end-state comfort posture [Weigelt et al., 2007]. This can be applied in robotics, for reaching motions as a check that the desired end-state is kinematically reachable, which provides an early detection of failure. The fact that a subject remembers the grasp pose can be applied in robotics as storing a successful grasp pose for later re-execution. Similarly, when a grasp fails the corresponding pose might be discarded or enlisted as a "bad pose" for this grasp type or object.

For goal directed motions Harris and Wolpert [1998] suggested the *minimum variance* hypothesis, which incorporates several features of the human movement into one framework. The minimum variance hypothesis states that the variance of the end-effector position should be minimized over multiple trials. This results in smooth (minimum jerk) motions. The variance of a motion is a cost function during the post-movement period:

$$C_{variance} = \sum_{t=T+1}^{T+N-1} V(t) \tag{2.5}$$

where $V(t)$ is the positional variance at time t, T is the time at the goal state and N is the post-movement period. This hypothesis explains several observed properties of human reaching, including Fitt's law, two-third power law, and the bell shaped velocity profiles which all are characteristics of human motions.

Simmons and Demiris [2005] have implemented a minimum variance controller for a robotic arm. They used a planar fully actuated two link arm to perform reaching motion. They showed that the minimun variance model not only describes characteristics of human arm movements, but also allowed a robot move in a humanlike way.

Instead of applying a model that resembles humanlikeness, Rigotti et al. [2001] used a neural network trained from human motions from a motion capturing system to drive an avatar, i.e., a human model. This technique can be translated to control a robot manipulator, thus moving in a humanlike way. The advantage of such an approach is the use of real digitalized human motions to drive the model instead of applying a model derived from empirical data. By imitating the actual recorded motion the regenerated motion will preserve some human characteristics. As discussed is the introduction in chapter 1, to simply play back the recorded trajectory is inappropriate for several reasons: different structures (correspondence problem), noisy data and no possibility to

generalize to new situations. However, the correspondence problem (see section 2.1) need to be addressed, since humans and robots have different body configurations. They used a model of the human that was *designed* to fit the human morphology, thus, they could ignore the correspondence problem in their study.

2.3 Segmentation of Motion Trajectories

Since both empirical and theoretical evidence suggests that rhythmic motions are different from discrete motions [Huys et al., 2008], we consider only the latter one in this thesis. To reproduce a complete demonstration it is necessary to segment the demonstration into a set of discrete motions. Another important reason for having a segmentation process is to determine if a motion is goal-directed, i.e., in relation to some object or point. The reference frame can either be an absolute frame, a relative frame to some object (hand-state space), or to reproduce some motion pattern, for example, a gesture.

Fod et al. [2002] outlined and compared two techniques for automatic segmentation of recored motions. Four joint angles of a human subject were recorded and segmented. The first segmentation technique used the angular velocity of each joint to detect zero velocity crossings. The second segmentation technique used the sum of all measured angular velocities. A motion was then defined as the segment above a threshold.

Ehrenmann et al. [2002] used a sensor fusion approach for trajectory segmentation. A segmentation point in time is identified by analyzing the force from a force sensor (mounted on the fingertip) together with the position, velocity, and acceleration with respect to a minima.

Another simple and effective way to segment manipulation demonstrations is measuring the mean squared velocity of the demonstrator's end-effector [Skoglund et al., 2007]. This technique is equivalent to the one developed by Fod et al. [2002], but instead of using the mean squared velocity of the joints angles, the end-effector's velocity were used.

2.4 Imitation Levels

Imitation can occur at different levels. Robotics research on imitation learning can be put into four categories:

Task level Also called *symbolic level* or *high level*. The task level represents a skill as a sequence of symbolic actions, e.g., [Ekvall and Kragic, 2006] and [Pardowitz et al., 2007]. Typically this approach uses classification techniques to identify a set of predefined actions within a demonstration. The advantage is that once the sequence of actions is recognized in the demonstration the robot maps these to its own actions, which typically

are preprogrammed so that the robot actually can execute the task. However, if an action cannot be classified, it is usually hard to build a new action from the demonstration within this framework.

Trajectory level This refers to when the demonstrated trajectory is mapped into a robot trajectory, e.g., [Ijspeert et al., 2003]. This approach typically concerns the correspondence problem that is, how to perform the mapping from human to robot. New, previously unseen actions are encoded into new robot actions (or primitives). Using this approach it is hard to combine actions in a sequence and reason about in what order they should occur or what the intention of a particular movement is.

Goal level Means when the goal of an action is inferred from action observation to acquire knowledge of what task to perform [Cuijpers et al., 2006]. Typically, the robot has a priori knowledge on how to perform an action (as in the Task level approach).

Model-based level Is when a predictive model of the world is learned, such as model based reinforcement learning discussed in chapter 5. Schaal [1997] showed that this type of learning greatly benefits from demonstrations in contrast to other reinforcement learning approaches. This approach is related to "predictive forward models", discussed in section 2.2.1.

We will elaborate on the task- and trajectory levels of imitation since they are important to the subsequent chapters of this thesis. The goal based level is not considered in this thesis. In task encoding of a skill the segmentation of a demonstration is one of the most important steps. Each of the segments should be encoded into a predefined action. Each of these actions has a symbolic description (e.g., "grasp object"). Symbolic descriptions represent a task as sequence or graph where each node is a skill. This representation is the main advantage of a symbolic description. The main drawback is that symbolic descriptions require predefined actions, which instead would be more advantageous to learn, since situations might occur where the predefined actions cannot execute the task adequately.

A common approach to make an abstract representation of a task is to form a hierarchical tree [Aleotti et al., 2004, Dillmann, 2004]. A set of skills can execute the task at the top of the hierarcy. Each skill is in turn decomposed into a set of movement- or action-primitives, also called elementary actions or basic tasks. Ultimately, a finite set of well designed movement primitives should be able to generate an arbitrary skill.

Like other task-based approaches, Ekvall and Kragic [2006] viewed a demonstrated task as a composite of specific actions. In addition they also took the impact of an action onto the current world state, thus effectively combining the goal with the task level. For example, besides the action an object position was taken into account which enabled a similarity measure of effects on the

world state. An important point was to determine if the change of position of an object were absolute or relative by computing the minimum variance with respect to other objects:

$$relobj_i = \underset{\forall\, j\ moved}{argmin}\ |cov(x_i - x_j)| \tag{2.6}$$

where x_i is the position of object i.

For trajectory encoding several different methods has been proposed. They will only be briefly reviewed here since we use a different method: fuzzy modeling in combination with a next-state-planner.

From a stereo vision system Ude [1993] recorded demonstrated trajectories. The output from the vision processing was a list of discrete points describing the object's trajectory during a demonstration with a high level of noise. From the noisy data, splines were used leading to smooth trajectories that a simulated robot could follow.

From a set of demonstrated trajetories, Aleotti and Caselli [2006] used distance clustering to group those trajectories that correspond to "macro-movements" representing qualitatively different executions of the demonstration. The proposed technique is a combination of Hidden Markov Models (HMM) for trajectory selection and Non-Uniform Rational B-Splines (NURBS) for trajectory encoding. The HMM distance clustering prevents all demonstrations from being fused into one average model, thus risking obstacle collision.

By combining Gaussian Mixture Regression with Gaussian Mixture Models, Calinon et al. [2007] encoded both the trajectory and the variability (see section 2.5) into a single coding scheme. The trajectory encoding used mean vectors and covariance matrices at selected points. This results in smoother average trajectory, where all demonstrations are fused into an average.

To encode a demonstrated trajectory, Ijspeert et al. [2002] used a set of Gaussian kernel functions, where each kernel is:

$$\Psi_i = \exp(-\frac{1}{2\sigma_i^2}(\tilde{x} - c)^2) \tag{2.7}$$

where $\tilde{x}$ is the distance to the goal, σ is the width of the Gaussian distribution and c the position. Locally weighted learning was used to find the weights w_i which minimize the criterion $J_i = \sum_t \Psi_i^t(u_{des}^t - u_i^t)$, where u_{des} is the desired output. This output is the difference $u_{des} = \dot{y}_{demo} - z$, between the demonstrated trajectory y_{demo} and the internal state variable z.

Compared to the above approaches, Kuniyoshi et al. [2003] proposed a quite different trajectory encoding in a self-learning task. A robot explored different motion patterns and observed its own actions with a camera. An optical flow was generated with 12 directions for each of the 400 points from a 256-by-256 image. To encode the motions a set of Gaussians with binary activation represents the joint angles. The mapping from the visual input to the motor

output was fed to a *non-monotonic neural network* as a temporal sequence. A non-monotonic neural network can encode a temporal sequence instead of a static point, hence, the network can learn a trajectory. After the robot have leaned the vision-motor association, it could imitate a demonstrator's arm action by performing similar arm actions.

2.5 Performance Metrics in Imitation

One must take into account the nature of the skill to be learned, as Dawkins [2006, chapter 6, page 224] describes "digital" and "analog" skills distinguished by how they are performed. In *analog skills*, a trajectory is learned by making a copy of the teacher's motion. An example of an analog skill could be dancing where a specific goal is missing. In such a case, it is hard to formulate some evaluation metric other than the teacher's motion. In *digital skills*, the motion must have a specific goal, such as in a reaching task or hammering. The goal of a reaching task is to reach a point, despite the initial position of the hand. When hammering, the nail should be driven in despite how many hits it takes.

To the knowledge of the author, there is no study on how to *distinguish* digital from analog skills. The two kinds of skills need different evaluation methods. The objective of an analog skill (such as gestures) is to minimize the difference between the observed action and the executed action, an issue addressed by several authors including Kuniyoshi et al. [2003], Billard et al. [2004] and Lopes and Santos-Victor [2007]. For digital skills a different measure is needed, usually also including world state since these skills often are object and goal state dependent, addressed for example by Erlhagen et al. [2006] and Jansen and Belpaeme [2006].

It is possible to infer the metric of a skill, meaning the constraint, from several demonstrations with a slight variation. By applying some statistical method on the data set, the essential parts of the demonstration can be identified and constraints on the motion can be applied in this region while other regions with less or no constraints can vary.

To apply a *Jordan curve theorem* at least two demonstrations are needed [Delson and West, 1994]. From two demonstrations, it is possible to determine the outer boundaries in a two dimensional task. As new demonstrations add more data to the data set, the task boundaries might be extended. A drawback of this approach is if there are two possible ways to go around an obstacle, and the demonstrator shows both so that the obstacle is *within* the boundaries.

Calinon et al. [2007] made generalizations across several demonstrations by measuring variance and correlation information of joint angles, hand path and object-hand relation. To have access to all this information the robot was thought by kinesthesis, i.e., the teacher moves the limbs of the robot, while the robot records temporal and spatial information. This metric was then used to analytically find an optimal controller. The use of an average trajectory could

be problematic for obstacles (like with the above mentioned Jordan curve) if different types of demonstrations will be fused into one with a higher variance.

Pardowitz et al. [2007] fused several sensor modalities (voice besides from motions) and looked at the frequency that some features occur across several demonstrations to form a relevance measure for each feature. A weighting mechanism enables the system to cope with both sparse knowledge at the initial learning phase and adapts when more knowledge arrives through new demonstrations.

On a symbolic level Ekvall and Kragic [2006] builds a list of constraints describing in what order states should occur. Initially, all actions are constrained to the orders in which they were demonstrated. As more demonstrations are appended, the list is updated to remove contradicting constraints.

2.6 Imitation Space

Skills can encode human movements in different spaces: joint space [Fod et al., 2002], Cartesian space [Dillmann, 2004], torque space [Yamane and Nakamura, 2003], visual coordinates [Lopes and Santos-Victor, 2007] or and even a combination [Kuniyoshi et al., 2003]. Here, we introduce *hand-state space* and its application in PbD.

As a part of the MNS model, Oztop and Arbib [2002] introduced the concept of hand-state, which stems from research in neurophysiology, more specifically schema theory. A hand-state trajectory describes the evolution of the hand's pose and shape (finger configurations) in relation to the target object. That is, it encodes the goal-directed motion of the hand during reach-and-grasp. Thus, when a robot tries to imitate an observed grasp, it has to move its own hand so that it follows a hand-state trajectory similar to the one of the demonstration.

Recent experimental evidence by Ahmed et al. [2008] support the idea that the brain encodes motion in object coordinates, i.e., hand-state, as well as in Cartesian space. They suggest that the brain represents a new object first in an arm-centered space, but that this representation develops into the hand-state representation as the subject gains experience with the manipulated object.

The hand-state describes the configuration of the hand and its geometrical relation to the affordances of the target object. Thus, the hand-state H and its components are defined as:

$$\mathrm{H} = [\mathrm{h}_1 \ldots \mathrm{h}_{k-1},\ \mathrm{h}_k \ldots \mathrm{h}_p] \tag{2.8}$$

where h_1 to h_{k-1} are hand-specific components, and h_k to h_p are components describing the hand pose with respect to the object. The combination of hand-specific components and hand pose components distinguish the hand-state from other similar descriptions introduced by other authors such as [Ogawara et al., 2003, Liebermann and Breazeal, 2004, Ekvall, 2007, Calinon et al., 2007]. In

a PbD context Iliev et al. [2007] was the fist to consider the hand-state space and Skoglund et al. [2008] elaborated this work.

The formulation of the hand-state (equation 2.8) allows the representation of motions in terms of hand-state trajectories. Note that the hand-state components are specific to the hand's kinematic structure and the definition of the object affordances. This means that some hand-state components will be defined differently for different grasp types since they involve different object affordances. For robots imitating humans, we have to define H in such way that it matches the capabilities of particular types of end-effectors, e.g., dexterous robot hands, two-finger grippers, as well as the human hand.

We apply the hand-state concept in the work covered in chapter 4, which also describes the application in more detail.

2.7 Next-State-Planners

A next-state-planner (NSP) plans a motor action one step ahead given its current state. This is in contrast to traditional approaches for robots where the complete trajectory is pre-planned all the way to the target and then executed by the controller. The NSP approach is consistent with the equilibrium point hypothesis [Bizzi et al., 1995] in that a target position acts as an attractor point. The planner requires the current state to be known at each step as well as a control policy. The trajectory that the robot executes is generated by a control policy, commonly known as the dynamic system approach [Ijspeert et al., 2001, Iossifidis and Schöner, 2004, Tani et al., 2004, Erlhagen and Bicho, 2006, Hersch and Billard, 2008]. In this thesis we use the term "next-state-planner", as defined by Shadmehr and Wise [2005], because it emphasizes on the next-state-planning ability, and the alternative term "dynamic system" is very general.

One of the earliest next-state-planners is the VITE (Vector Integration To Endpoint), developed by Bullock and Grossberg [1989]. The VITE model is described by the two equations:

$$\begin{aligned} \ddot{x}(t+1) &= \alpha(-\dot{x}(t) + x_g - x(t)) \\ \dot{x}(t+1) &= \dot{x}(t) + \ddot{x}(t) \end{aligned} \tag{2.9}$$

where $\ddot{x}$ is the acceleration, α a positive constant and $x_g - x$ a difference vector between the goal and the current state, where the position of the end-effector at time t is given by x and the goal position is x_g.

Hersch and Billard [2008] proposed a combined controller with two VITE controllers running in parallel, to control a humanoid's reaching motion, with one controller acting in joint space and the other one in Cartesian space. Hersch et al. [2006] applied the VITE inspired controllers in a PbD framework where a humanoid robot executed a "put an object in a box" task. The data was

recorded using kinesthetic demonstrations. To weight the demonstrated trajectory against the trajectory produced by the next-state-planner a time dependent factor $\gamma(t)$ was used:

$$\gamma(t) = \gamma_0 max(((T-t)/T)^2, 0) \tag{2.10}$$

where T is the duration of the observed motion and γ_0 is a gain. The desired velocity $\dot{x}^d$ was determined by:

$$\dot{x}^d(t+1) = \gamma(t)\dot{x}(t) + (\gamma_0 - \gamma(t))\dot{x}^o(t) \tag{2.11}$$

where $\dot{x}^o$ is the demonstrated velocity and $\dot{x}$ is the actual velocity.

Several alternative NSPs have been suggested for different purposes. Ijspeert used two different dynamical systems as NSPs, one for a rhythmic motion and one for a discrete reaching task [Ijspeert et al., 2002, 2003].

Ito et al. [2006] introduced the recurrent neural network with parameter basis (RNNPB) where a dynamical system–acting as a NSP–is learned by the network. The network learned several behaviors through kinesthetic demonstrations, where the network encoded different behaviors using the parameter biases. This was tested for a humanoid robot with two 4 DOFs arms in a ball handling task, where the behaviors switched between rolling and lifting the ball.

2.8 Other Approaches to Imitation

There are other approaches to imitation learning, which deserve a brief explanation. They have not been explicitly addressed in this chapter, since they have little connection to the rest of this thesis.

Criticism to methods relaying on several demonstrations states that it is inadequate for the user and causes fatigue. In contrast, Friedrich et al. [1996] and Aleotti [2006] used *one-shot learning*. One-short-learning usually requires prior knowledge of the task, since without such knowledge it is hard to estimate essential parts of the demonstration which can lead to ambiguities [Ogawara et al., 2003]. It is desirable that there should be no prior knowledge and the required knowledge should be acquired from a demonstration. Thus, several demonstrations are typically needed.

Incremental learning is a unification of one-shot learning from multiple demonstrations in that it starts learning from the first demonstration and then continues. This leads to a learning process that gradually improves as more and more demonstrations are observed.

If the demonstration and the actions executed by the robot can be represented in some common domain, such as the hand-state space, and the robot has access to a performance metric of the task, then the robot can self-improve. This means that initially the learning agent watches the teacher performing the

task, and after some time the agent starts to perform the task itself. To improve the model, the agent evaluates its own performance after the initial trials in the same way as the teacher is evaluated. This approach is illustrated by Bentivegna et al. [2004], who conducted experiments with a humanoid robot that learned to play air hockey. The task was represented using a predefined set of movement primitives, that could both be observed in the human demonstration and executed by the robot. As a learning mechanism *locally weighted learning* was used to first learn from observing the teacher and then from self-observation. This approach falls under the model-based learning category as a model of the environment is learned.

2.9 Summary and Proposed Improvements

This chapter has reviewed PbD systems, their general functions and discussed drawbacks of existing methods. We have reviewed the key problems in imitation learning and have also discussed how to address them. We have identified topics of special interest: *how to map* a human demonstration to a robot; and *what* to imitate. We have also covered the subject of evaluating an imitation in terms of a performance metric. Characteristics of human motions, i.e., how to generate motions that are humanlike, have also been discussed and how they can be used as a performance metric. Typically, robotic imitation occurs in one of two levels: task or trajectory (high level or low level). One of the open issues in robotic imitation is how to combine these two levels within a single framework.

To address the identified problems and drawbacks of existing methods the rest of this thesis contain proposed improvements corresponding to the following list:

- A reversed learning algorithm to learn a model-free mapping from human to robot, explained in chapter 3.
- Use of recorded human motions to drive a robot in a humanlike way discussed in chapter 3 and 4.
- Application of a mathematical model that describes some characteristics of human motions (Fitts' law), chapter 4.
- As a novel approach of encoding demonstrated trajectories, fuzzy modeling is presented in chapter 3.
- A next-state-planner, based on the fuzzy modeling approach is used to execute the imitated task, discussed in detail in chapter 4.
- To provide a link between the human and the robot in the PbD framework, the hand-state notion is applied, discussed in chapter 4.

- How demonstrations can speed up reinforcement learning for a reaching task, discussed in chapter 5.
- How demonstrated skills can become new skills in the action repertoire of the robot using the hand-state space and minimum jerk as a performance metric, discussed in chapter 5.

Chapter 3
Supervised Learning in Programming-by-Demonstration

In this chapter, supervised learning methods are introduced and applied to position teaching by demonstration as well as task learning from demonstration using skills. Position teaching by demonstration deals with supervised learning of robot positioning with the help of a human a demonstrator. The approach presented in section 3.2 is different from most work on Programming-by-Demonstration (PbD) in that the human first imitates the robot, so that the robot can build a user specific model for positioning the end-effector. The method does not require any analytical modeling of neither the human nor the robot and can be customized for different users and robots. An experiment using this approach is presented in section 3.2.3. The setup consists of a wearable motion capturing device and a light weight manipulator.

A method for task learning from demonstration using skills is described in section 3.3. Skills are basic actions of the robot/gripper, which the robot/gripper can execute in a sequence to form a complete task. It includes how a demonstration of a pick-and-place task could be interpreted and mapped to a set of predefined basic skills (in our previous work referred to as "task primitives"). The demonstration is interpreted by a set of skills for a known type of task, then a robot program is automatically generated. A pick-and-place task is analyzed based on the velocity profile, and decomposed into skills. To illustrate the approach, we carried out an experiment using an industrial manipulator equipped with a vacuum gripper. The demonstration is captured using a magnetic tracker and a motion capturing glove, and modeled using fuzzy time-modeling.

Fuzzy time-modeling is based on Takagi-Sugeno modeling which we use to model the demonstrated trajectories and to generate robot trajectories. In section 3.1.3 Takagi-Sugeno fuzzy-modeling is introduced. We show that this modeling produces very compact models and results in smooth and accurate motion trajectories.

The contributions presented in this chapter are:

1. A method for *model free* mapping of positions from a human demonstration to a robot manipulator. "Model free" means that no knowledge of the demonstrator's arm configuration is required.

2. The introduction of *time-clustering* for modeling trajectories captured from human demonstration.

3.1 Supervised Learning

Supervised learning includes techniques that provide a learning agent with information on how to perform a certain task. Supervised learning requires some-kind of teacher, providing the agent with a target output vector $\mathbf{y}_i$, which is the desired output, for each input vector $\mathbf{x}_i$, where $i \in \{1, 2, \ldots, n\}$. The agent's objective is to form an input-output mapping $\hat{\mathbf{y}} = F(\mathbf{x})$, where $\hat{\mathbf{y}}$ is the predicted output, which minimizes the error E for a given training set. E is the sum of prediction errors:

$$E = \sum \|\hat{\mathbf{y}} - F(\mathbf{x})\| \tag{3.1}$$

To confirm that the agent has been trained to perform the intended task a test set is provided for validation. The test set is a data set different from the training set.

3.1.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are among the most well-known and well-used learning methods that were developed to imitate the function of neurons in the brain. An ANN consists of computing neurons and connections between them with weights assigned to each connection. This is true for all types of ANNs. However, the similarities between different types of networks ends there. Commonly used networks like a multilayer feedforward network, do not always provide the solution to the problem at hand. Hence, much research has been done on different network structures during the last few decades. Research on neural networks has resulted in numerous different types, each with its own characteristics and features. Multilayer feedforward neural networks (MLFF) are among the most commonly used ones.

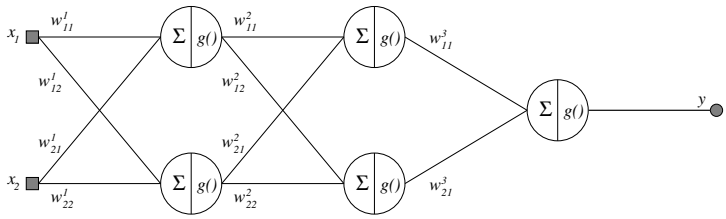


Figure 3.1: *A multilayer feedforward neural network with one input layer (usually not counted) two layers of perceptions, and one output layer with only one unit. This type of network is usually named "two layer MLFF neural network", because it has two layers of operating units.*

Multilayer Feedforward Networks

In the fully connected multilayer feedforward neural network, each input unit is connected to all neurons in the next layer. The layer without any direct connections to the input or output is called the hidden layer. All connections have an associated weight to it, a value usually between -1 and 1, where a value close to 0 indicates a weak connection, and a value close to -1 or 1 indicates either a strong negative or positive connection respectively. First, each neuron sums all the inputs multiplied by their respective weights. The weight for each connection is denoted by w^k_{ij} where $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, \ldots, m\}$, n is the number of input nodes and m the number of output nodes for the kth layer, with $k \in \{1, 2, \ldots, l\}$. Typically, k is a small number, e.g., 1 or 2. The sum is then:

$$s^k_i = \sum_j w^k_{ij} x^k_i + w^k_{0j} \tag{3.2}$$

where w^k_{0j} is a bias term. Then, an activation function denoted by $g()$ is used, also called a transfer function or squashing function, typically a sigmoidal function, often implemented by:

$$g_i(s_i) = \frac{1}{1 + e^{-s_i}} \tag{3.3}$$

or:

$$g_i(s_i) = \tanh(s_i) \tag{3.4}$$

which both produce a smooth step output around $s_i = 0$, from 0 to 1 and -1 to 1, respectively. Other common types of activation functions are step functions

and linear functions. Figure 3.1 shows a feedforward network with two input units with the input values x_1 and x_2. Each layer is first computed according to equation 3.2, and then the output of each neuron is calculated by a transfer function (equation 3.3 or 3.4). The output of a neuron becomes the input to the next layer.

To train the network, i.e., to tune the weights so they minimize the error, the most commonly used method is the backpropagation algorithm. In this algorithm, the error is back-propagated layer by layer and the weights are adjusted to decrease the error at each iteration of training.

3.1.2 Memory-Based Learning

Memory based learning (MBL), also known as "instance based learning" or "lazy learning", can be used for function approximation, summarized by Mitchell [1997, chapter 8]. Compared to other learning methods, for example, the multilayer feedforward network described earlier, the memory-based techniques generally do not require an explicit training step. Instead, they simply store each given training example. The absence of an explicit training phase has made MBL methods fast in training time, but slow on query time, since the generalization is postponed until a query is given. This is why these methods are often called "lazy". The names "instance based" and "memory-based" refer to the fact that all the training samples are stored. Memory-based methods, in general, have the property of never unlearning. A problem that most MBL methods share is that the query time grows as the number of data points increases.

Minimum Distance Classifier

Using a Minimum Distance Classifier method; the mean vectors are calculated for each class in the training data. The squared Euclidean distance to each of the mean vectors is calculated to classify a new input vector, and the vector is assigned to the class with the shortest distance.

Equivalently, the decision function for a minimum distance classifier can be written as:

$$d_j(\mathbf{x}) = \mathbf{x}^T \mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^T \mathbf{m}_j \tag{3.5}$$

where $\mathbf{x}$ is the pattern vector to be classified, and $\mathbf{m}_j$ is the mean vector for class j. Classification is then determined by the class that produces the highest decision value.

Distance Weighted Nearest Neighbour

The distance weighted nearest neighbor (DWNN) method is one of the most fundamental MBL methods, which assigns a weight to all training points based on a distance measure, for example, the Euclidean distance:

$$w = d_E(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_j (\mathbf{x}_j - \mathbf{q}_j)^2} \tag{3.6}$$

where $\mathbf{q}$ is the query point, $\mathbf{x}$ are the training vectors, and j is the number of elements in the vectors $\mathbf{x}$ and $\mathbf{q}$. This means that each data point has a weight, denoted w, based on the distance, which measures its contribution to the prediction. All selected points are then combined in a weighted sum to form the prediction:

$$\hat{\mathbf{y}} = \frac{\sum_{i=1}^{k} w_i \mathbf{x}_i}{\sum_{i=1}^{k} w_i} \tag{3.7}$$

where $\hat{\mathbf{y}}$ is the predicted output, and k is the number of training vectors.

Locally Weighted Learning

A further development of the distance weighted nearest neighbour method is locally weighted learning (LWL), surveyed by Atkeson et al. [1997a,b]. Like DWNN, LWL is a memory based method but instead of just using the distance measure a weighting kernel function also included that require tuning, making it a more sophisticated algorithm.

LWL has shown to scale well to problems with high dimensionality [Schaal et al., 2001, 2002]. In many real world examples the number of dimensions is very high, however, only small fractions contain useful data. Vijayakumar et al. [2005] interpret this as evidence that, despite the often high number of variables, the real dimension is lower. This can be illustrated by an example: when looking at one pixel in a picture of a natural scene, the surrounding pixels are often very similar. That is, the picture contains redundant information, and therefore the picture can be compressed. The same argument is also valid for the human body posture.

LWL is very well suited as a function approximator for reinforcement learning, which means an approximation of the output, a scalar value denoted by y, with an approximation $\hat{y}$, based on known input vectors $\mathbf{x}$. The inputs $\mathbf{x}$, can be of higher dimension, denoted by the bold notation $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, while the output is a scalar, denoted $\{y_1, y_2, \ldots, y_n\}$, and n is the number of training samples.

If different dimensions have different measures they might have to be scaled. This scaling is necessary, for example, when an approximation of both states and actions are needed, but the states and actions are of different measures. For example, the state could be the individual joint angles of a manipulator while the actions are the joint's velocity. A distance measure is applied to assign different weights to different dimensions, which are known as scaling the query

point $\mathbf{q}$. An important note is that this is done for *each input dimension*. Different distance functions can be used for this scaling, for example, the unweighted Euclidean distance, equation 3.6, or the diagonally weighted Euclidean distance:

$$d_m(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_j (m_j(\mathbf{x}_j - \mathbf{q}_j))^2} \tag{3.8}$$

Using a scaling factor of zero does not mean that dimensions are ignored in LWL. For dimensionality reduction principal component analysis or singular value decomposition should be used instead.

Each data point is then *weighted* using a kernel function, denoted by $K()$, that uses the result from the distance function $d(\mathbf{x}_i, \mathbf{q})$. This is similar to the weighting function earlier described in the DWNN method. There are several different options when choosing a kernel function for the weighting, such as: distance to a negative power, $K(d) = \frac{1}{d^p}$, Gaussian kernel, $K(d) = e^{-d^2}$ etc. The output prediction is then formed by:

$$\hat{y}(\mathbf{q}) = \frac{\sum y_i K(d(\mathbf{x}_i, \mathbf{q}))}{\sum K(d(\mathbf{x}_i, \mathbf{q}))} \tag{3.9}$$

Again, $d()$ is the distance metric, $K(d)$ is the kernel function and y_i the output for the query point.

Locally Weighted Regression Projection

Locally Weighted Regression Projection (LWPR), developed by Vijayakumar et al. [2005], is an extension of the LWL algorithm in which LWPR does not explicitly store training data, hence, is not a memory based algorithm anymore. In LWPR statistics are used instead of storing all the data points, so the learning is accumulated in statistical variables. This makes LWPR well suited for incremental learning. A forgetting factor is also incorporated into the algorithm to forget older data, another important feature of incremental learning strategies. Another advantage LWPR has over LWL is the automatic distance metric adaptation, instead of the a fixed distance metric D.

3.1.3 Fuzzy Modeling

Takagi and Sugeno [1985] proposed a structure for fuzzy modeling of input-output data of dynamical systems. Let $\mathbf{X}$ be the input data set and $\mathbf{Y}$ be the output data set of the system with their elements $x \in \mathbf{X}$ and $y \in \mathbf{Y}$. The fuzzy model is composed of a set of c rules R from which rule R_i reads:

$$\text{Rule } i: \text{ IF } x \text{ IS } \mathbf{X}_i \text{ THEN } y = A_i x + B_i \tag{3.10}$$

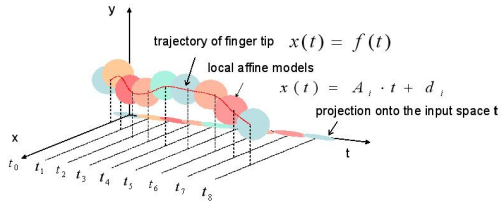


Figure 3.2: *Time-clustering principle.*

$\mathbf{X}_i$ denotes the i:th fuzzy region in the fuzzy state space. Each fuzzy region $\mathbf{X}_i$ is defussified by a fuzzy set $\int w_{x_i}(x)|x$ of a standard triangular, trapezoidal, or bell shaped type. $W_i \in \mathbf{X}_i$ denotes the fuzzy value that x takes in the i:th fuzzy region $\mathbf{X}_i$. A_i and B_i are fixed parameters of the local linear equation on the right hand side of equation 3.10.

The variable $w_i(x)$ is also called degree of membership of x in $\mathbf{X}_i$. The output from rule i is the computed by:

$$y = w_i(x)(A_i x + B_i) \tag{3.11}$$

A composition of all rules $R_1 \ldots R_c$ results in a summation over all outputs from equation 3.11:

$$y = \sum_{i=1}^{c} w_i(x)(A_i x + B_i) \tag{3.12}$$

where $w_i(x) \in [0, 1]$ and $\sum_{i=1}^{c} w_i(x) = 1$.

The fuzzy region $\mathbf{X}_i$ and the membership function w_i can be determined in advance by design or by an appropriate clustering method for the input-output data. In our case we used a clustering method to cope with the different non linear characteristics of input-output data-sets (see [Gustafson and Kessel, 1979] and [Palm and Stutz, 2003]). For more details about fuzzy systems see [Palm et al., 1997].

In order to model time dependent trajectories x(t) using fuzzy modeling, the time instants t take the place of the input variable and the corresponding points x(t) in the state space becomes the outputs of the model.

For the purpose of modeling human demonstrations, e.g., end-effector trajectories, the incorporation of the time in the modeling process has the following advantages:

1. Representation of the dynamic arm motion by a small number of local linear models and few parameters.

2. Nonlinear filtering of noisy trajectories.

3. A simple interpolation between data samples.

The Takagi-Sugeno (TS) fuzzy model is constructed from captured data from the end-effector trajectory described by the nonlinear function:

$$\mathbf{x}(t) = \mathbf{f}(t) \tag{3.13}$$

where $\mathbf{x}(t) \in \mathbb{R}^3, \mathbf{f} \in \mathbb{R}^3$, and $t \in \mathbb{R}^+$.

Equation (3.13) is linearized at selected time points t_i with

$$\mathbf{x}(t) = \mathbf{x}(t_i) + \frac{\Delta \mathbf{f}(t)}{\Delta t}|_{t_i} \cdot (t - t_i) \tag{3.14}$$

resulting in a locally linear equation in t.

$$\mathbf{x}(t) = \mathbf{A}_i \cdot t + \mathbf{d}_i \tag{3.15}$$

where $\mathbf{A}_i = \frac{\Delta \mathbf{f}(t)}{\Delta t}|_{t_i} \in \mathbb{R}^3$ and $\mathbf{d}_i = \mathbf{x}(t_i) - \frac{\Delta \mathbf{f}(t)}{\Delta t}|_{t_i} \cdot t_i \in \mathbb{R}^3$. Using (3.15) as a local linear model one can express (3.13) in terms of an interpolation between several local linear models by applying Takagi-Sugeno fuzzy modeling [Takagi and Sugeno, 1985] (see figure 3.2)

$$\mathbf{x}(t) = \sum_{i=1}^{c} w_i(t) \cdot (\mathbf{A}_i \cdot t + \mathbf{d}_i) \tag{3.16}$$

$w_i(t) \in [0, 1]$ is the degree of membership of the time point t to a cluster with the cluster center t_i, c is number of clusters, and $\sum_{i=1}^{c} w_i(t) = 1$.

The degree of membership $w_i(t)$ of an input data point t to an input cluster C_i is determined by

$$w_i(t) = \frac{1}{\sum_{j=1}^{c} \left(\frac{(t-t_i)^T M_{i\,pro}(t-t_i)}{(t-t_j)^T M_{j\,pro}(t-t_j)} \right)^{\frac{1}{\tilde{m}_{proj-1}}}} \tag{3.17}$$

The projected cluster centers t_i and the induced matrices $M_{i\,pro}$ define the input clusters C_i ($i = 1 \ldots c$). The parameter $\tilde{m}_{pro} > 1$ determines the fuzziness of an individual cluster [Gustafson and Kessel, 1979].

3.2 Position Teaching of a Manipulator

In this section we will present a position teaching approach, where human arm positions are mapped to robot arm positions by learning the correspondence. Different robotic manipulators have different configurations and kinematic constraints. Most of them do not permit a straightforward mapping from the human arm domain to the robot's restricted joint space. In this experiment the aim is to provide a general method for mapping human motions to the robotic arm domain, assuming a redundant human arm will teach a robot arm with few degrees of freedom (DOF). Data capture is achieved with the special

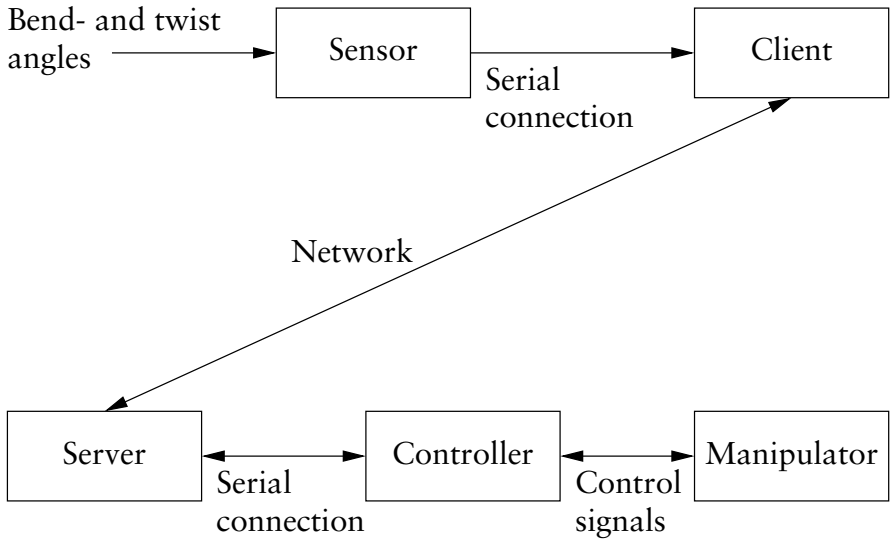


Figure 3.3: *A simplified schematic picture of the experiment setup.*

motion capture sensor called ShapeTape™ that is worn by the human operator (see section 3.2.1). The method is general in the sense that any human arm (or even leg, neck, or spine) could be mapped onto any robotic manipulator.

With this approach, the mapping from human to robotic joint angles need not be isomorphic (one-to-one). The method permits a human demonstrator to control non-anthropomorphic robots and to decide which arm motions to associate with different robot movements (controls). Non-linear function approximation algorithms such as artificial neural networks are used to learn a direct sensor-motor coupling from the sensor readings to the desired joint angles of the robot. We also propose an interactive training method, whereby the robot and human take turns to follow each others' actions until the learned mapping is considered satisfactory by the user. Experiments with a real manipulator are presented in section 3.2.3.

3.2.1 Experimental Setup

The experimental setup consists of a motion capture sensor, a robot manipulator, and two workstations running Windows. Figure 3.3 provides a schematic overview of the setup.

The robotic system comprises a light-weight arm called PANDI-1, shown in figure 3.4, developed "in house" at the AASS Intelligent Control Laboratory [Ananiev et al., 2002]. It has six degrees of freedom and can lift a maximum

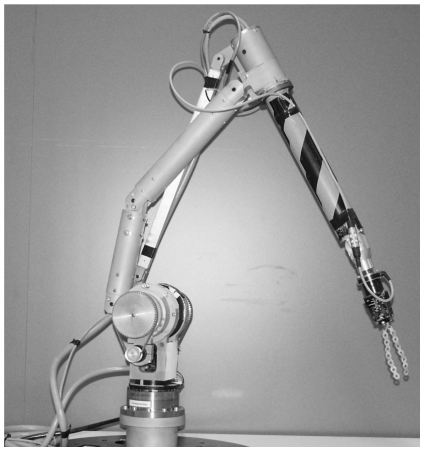


Figure 3.4: *The light weight robot manipulator Pandi-1 used in the experiments.*

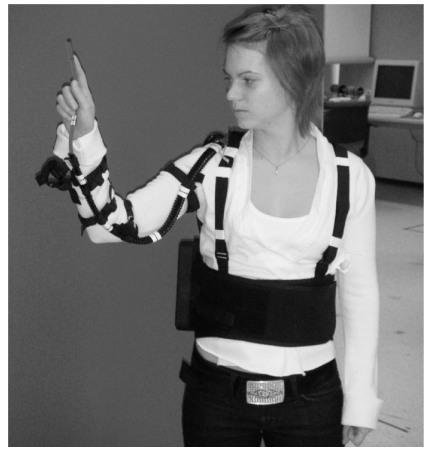


Figure 3.5: *The ShapeTape sensor attached to a demonstrator.*

payload of 3.5 kg (nominal 2 kg) and weighs about 18 kg, including controller and power electronics. Only the first three joints were used in this experiment.

The platform has a client-server structure shown in figure 3.3. From the client side the user can choose between different options: to collect the data for the training phase, to train and test the learning algorithms or to control the robot manipulator in real-time with the trained system.

The motion capture sensor used for collecting the human arm data is the ShapeTape from Measurand Inc. shown in figure 3.5. The ShapeTape, a multi degree of freedom input device, is a 1.84 m long strip containing optical light sensors that measure the bend and twist angles between two sensor segments, at 110 Hz. The sensor provides curvature data in vector format: the data are paired in bend and twist angles for 16 distinct positions along its length. The demonstrator attaches the tape to the arm with straps. A part of the sensor is looped onto the back of the demonstrator to fit the tape to the arm.

3.2.2 Methods

We propose a general training strategy where the robot learns a user model to map user-to-robot positioning. The method is based on learning a direct sensor-motor coupling from the ShapeTape captured data to the joint space of the robot. Mapping the sensor readings to the desired joint angles is non-trivial because the input state space is large and the unknown target function is non-linear. This gives us a "2m-to-n" mapping problem which could be illustrated in the form:

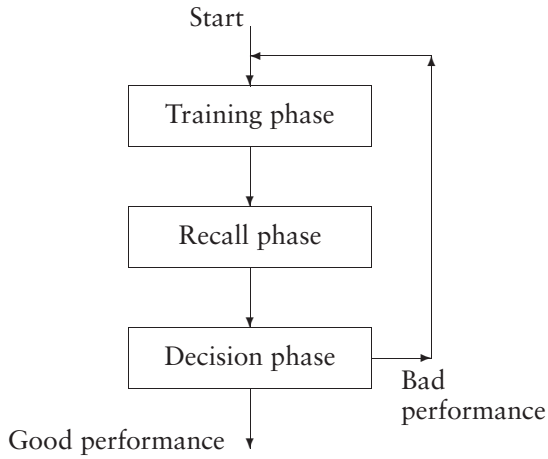


Figure 3.6: *A flowchart of the interactive learning process.*

$$\begin{bmatrix} b_1 \\ t_1 \\ b_2 \\ t_2 \\ \dots \\ b_m \\ t_m \end{bmatrix} \Rightarrow \begin{matrix} \text{Learning} \\ \text{Algorithm} \end{matrix} \Rightarrow \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \dots \\ \theta_n \end{bmatrix}$$

where b_m and t_m are the bend and twist angles respectively for each of the m sensors and θ_n is the angle for each of the n joints. In practice, as stated before, not all of the sensor values from the ShapeTape can be used (only those which are actually located along the human arm). In the experiments presented here, we use $m = 7$ and $n = 3$.

It would be advantageous to learn the mapping from human to robot, since this mapping (using this type of sensor) will be different form human to human. This means that different sections of the sensor will correspond to the same manipulator joint for different human demonstrators. The performance of two different learning algorithms was investigated, namely, a minimum distance classifier (MDC) and a multi-layer feed-forward (MLFF) neural network.

Interactive Learning

The user interface allows the demonstrator to teach the system with an interactive learning procedure. The system informs the demonstrator and asks for feedback after each cycle in the interactive learning loop, shown in figure 3.6. We divide the interactive cycle into three different phases: the *training phase*,

the *recall phase* and the *decision phase*. The system goes through each phase as follows:

Training phase. The robot makes a sequence of motions and the human demonstrator copies them. Then, the recorded data from the sensor, and the manipulator is used for training and testing the selected learning algorithm.

Recall phase. The human demonstrator controls the robot in real-time by moving the arm and the robot perform the corresponding commands by using the learned behavior.

Decision phase. The human demonstrator decides whether the resulting motion was satisfactory.

These phases are repeated until the human demonstrator is satisfied with the result (this is a subjective decision made by the user).

In the first phase, the robot is programmed to perform a sequence of small movements and, after each movement, the user is asked to copy the observed motion by moving the hand (and thereby affecting the measured joint angles) to the corresponding final position in the user's workspace. The user is free to choose which motions to associate with different robot joints movements. For each movement the system records 100 sensory readings and then computes the average for each sensor of the ShapeTape. This process takes approximately one second while the user is asked to leave the arm in a fixed position. This approach tries to compensate for small variations in the position of the arm and noise on the sensor readings. After the data collection procedure the sensory readings are preprocessed to get transformed variables with zero mean and unit standard deviation.

In the recall phase, the system cycles through a loop that consists of several steps. First, a new input pattern is read from the ShapeTape sensor that corresponds to the current configuration of the demonstrator's arm. These raw data are then preprocessed and used as inputs to the trained system, which computes the output joint angles. The outputs are discarded if they exceed the joints' limits or if the difference between each new computed angle and the previous one exceeds a predefined threshold for safety reasons. Otherwise, the movement command is sent to the server that controls the robot arm.

In the decision phase, the demonstrator decides whether the robot's motion was satisfactory or not according to the error between the resulting performance and the expected behavior. If the demonstrator accepts the learned policy then the interactive learning process ends. The parameters of the learning algorithms are saved and can be reused by the same demonstrator in a future session.

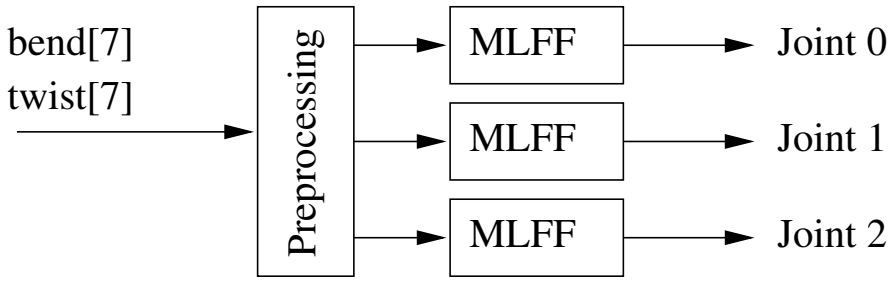


Figure 3.7: *Ensemble of MLFF networks.*

Minimum Distance Classifier

The first algorithm that was implemented is based on a minimum distance classifier (MDC) [Duda et al., 2000].

In the implementation the classes are points organized in a grid, and each point is associated with a vector of bend and twist data stored in a look-up table.

The MDC was successfully tested on a two-joint workspace for the manipulator. During the data collection phase the manipulator is programmed to reach each vertex in its workspace, and the user is asked to copy the robot by moving the hand to a corresponding position. The robot cycles several times through the grid to collect enough data for the training and testing phases. In the training phase the look-up table is built by simply associating the preprocessed sensor data, read from one loop, with the corresponding joint angles of the manipulator.

During the test- and recall-phases the MDC finds the point in the look-up table with the minimum squared distance from the actual input vector. To produce a continuous function approximation from the sensory inputs to the desired outputs a linear interpolation is applied between the nearest vertex and its neighbors in the vertical and horizontal directions.

While this learning algorithm provided a useful proof of concept, generalization of this approach to higher dimensional grids would be tedious and time-consuming (resulting from the well-known "curse of dimensionality"). We therefore investigated the generalization capabilities of artificial neural networks, described as follows.

Ensemble of Multi-layer Feedforward Networks

The second learning algorithm that was integrated into the system is based on an ensemble of multi-layer feed forward (MLFF) neural networks [Sharkey, 1996]. One MLFF network is trained for each of the robot's joint angles. Fig-

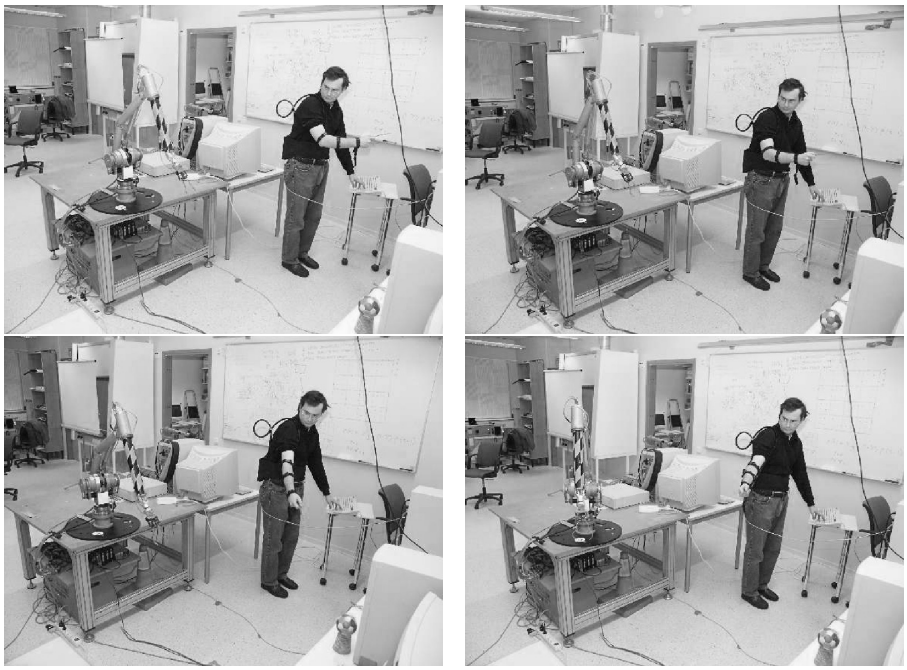


Figure 3.8: *A sequence of images showing the recall phase.*

ure 3.7 shows the structure of the neural network ensemble. The standard MLFF architecture was applied; using sigmoid activation functions in the hidden units and linear activation functions in the output units (described in section 3.1.1).

The $2 \times m$ raw input data are first preprocessed, then the transformed variables are used as input to the three independent MLFF networks, one for each used joint of the manipulator. An alternative solution with only one neural network with three outputs was also investigated as a reference.

The parameters that can be changed in the training phase include the number of hidden layers and units, the number of training epochs, and the error tolerance for the backpropagation algorithm. In the experiments the number of hidden units was determined manually by validation on the test data, but an automatic procedure for adding and pruning hidden units could also be used, such as the cascade-correlation algorithm [Fahlman and Lebiere, 1990]. After the training process each network generates a weight file that is used for the test and recall phases.

	Algorithm	No. joints	SSE	Mean Error(deg)
a	MDC (9×9 grid)	2	2.95	9.00
b	MDC (5×5 grid)	2	5.43	23.45
c	1 MLFF	2	8.20	15.00
d	2 MLFF	2	2.40	8.13
e	3 MLFF	3	3.60	8.13

Table 3.1: *Experimental results from the test phase.*

3.2.3 Experimental Evaluation

Our first experiment was a simple reaching task where the human demonstrator tries to lead the robot manipulator to a specific point in space or to follow a sequence of movements. The two learning algorithms were evaluated on tasks that involve the first two joints of the manipulator. Table 3.1 summarize the results: the chosen performance measures are the sum squared error (SSE) and the mean error per joint expressed in degrees on an independent test set.

The data set in the robot's workspace consists of a grid of 81 points (9×9) equally spaced in the horizontal direction by 20 degrees and in the vertical direction by 10 degrees. Thus, the resulting grid lies on the surface of a sphere. The demonstrator was asked to copy the robot's movements, producing 81 corresponding configurations of the arm, and to repeat the whole sequence 3 times (thus the total size of the data set was 243 examples). For the training phase, one group of 81 examples was chosen for both the MDC and MLFF algorithms, while the remaining examples were used for testing. In experiment b the MDC was trained with a smaller data set of 25 points to investigate how the performance degrades if the resolution of the grid decreases.

The best result with the lowest SSE of 2.4 (experiment d) was obtained with 2 neural networks, each one controlling a specific joint. The corresponding mean error was 9 degrees for each joint. Figure 3.9 shows the expected and output angles of the first joint for a sequence of patterns (experiment d). Experiment c comprised a single neural network with two outputs: the resulting mean error is quite large and almost 80% of the error was caused by the second joint. This demonstrates that the solution with only one neural network is not effective.

In the experiments, the MLFF networks were trained with non-linear sigmoid activation functions in the hidden layers and linear activation functions in the output layers. The size of the hidden layer was fixed to 4, based on the SSE on the test data.

The reaching task was also extended to learn the position of three active joints using the ensemble of MLFF networks (case e). The system was trained in free space by collecting a sequence of 243 examples with multiple demon-

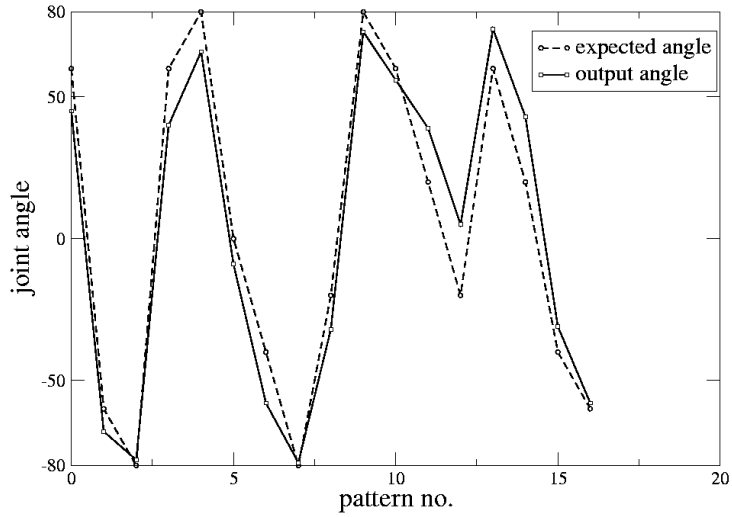


Figure 3.9: *Expected angle and output angle for the base joint 1 (16 test patterns).*

strations. After training, the performance of the system was validated in the recall phase. The robot arm could follow the user's actions as shown in the sequence of pictures in figure 3.8.

After the training phase the user remarked that it is hard to show the robot the same position several times, because reaching a point in free space with the same configuration of the joints is non-trivial without some reference. This is probably the main source of error in the current system.

We have also tried some more complex experiments that involve grasping of objects. While moving the arm, the demonstrator can also open or close the end-effector of the robot by pressing a keyboard. The performance was not satisfactory because the trained algorithm produces a position error. This error often leads to an insufficient positioning accuracy of the gripper.

3.2.4 Discussion

We have described a system for robot-learning-from-demonstration which is able to capture and map a human control policy to a robot arm. We show how a motion capture sensor can be used to train a learning algorithm to position a robot manipulator. Artificial neural networks and a minimum distance classifier were tested, where an ensemble of multi-layer feed forward neural networks (MLFF) performed slightly better than the minimum distance classifier. The method provides the robot with a model of the human operator and a mapping from a particular human pose to a corresponding robot pose. The experiment showed that the performance was not satisfactory for precise reach-and-grasp

tasks. The main cause of discrepancies in the current system is probably the use of absolute bend and twist angles in the input to the learning algorithm, since it is very difficult for a human demonstrator to position the arm at exactly the same configuration in repeated trials. Since the measurement is relative to the base of the sensor, an additional sensor is required to localize the base. In the case of MLFF, a source of error can also be the separation of the networks. Since the joints are coupled, the networks shouldn't be treated individually.

3.3 Task Learning from Demonstration Using Skills

In this section, we describe a method for task learning from demonstration using skills. We focus on a pick-and-place task using a 6 degree-of-freedom manipulator and a vacuum gripper. The type of task is assumed to be known and mapped on to a set of predefined skills. A task is composed of a sequence of skills. The skills in our approach are designed for a particular gripper type, reducing the risk for failures during the execution phase and misunderstanding during the demonstration phase. Thus, our approach alleviates the correspondence problem [Nehaniv and Dautenhahn, 2002]. In addition, we assume that the demonstrator is aware of the functionality of the particular robot, and its skills directly reflect specific capabilities of the robot arm/gripper. In this way, the skills serve as a common language for the robot and the demonstrator. The sensor system consists of a magnetic tracker–mounted on the hand of the demonstrator–to record the demonstrated trajectories. Another benefit of our setup is that we do the teaching in the manipulator's own coordinate system, where the manipulator and the demonstrator share the same workspace.

3.3.1 Skill Encoding using Fuzzy Modeling

Five important desirable properties for encoding movements have been identified by Ijspeert et al. [2001]. These are:

1. The representation and learning of a goal trajectory should be simple.
2. The representation should be compact (preferably parameterized).
3. The representation should be reusable for similar settings without a new time consuming learning process.
4. For recognition purpose, it should be easy to categorize the movement.
5. The representation should be able to act in a dynamic environment and be robust to perturbations.

Let us examine the properties of fuzzy modeling with respect to the above enumerated desired properties. Fuzzy modeling is simple to use for trajectory learning and is a compact representation in form of a set of weights, gains and

offsets (i.e., they fulfill property 1 and 2) [Palm and Iliev, 2006]. To change a learned trajectory into a new one for a similar task with preserved characteristics of a motion, Iliev et al. [2007] proposed an algorithm using fuzzy time modeling, thus addressing property 3. Furthermore, the method satisfies property 4, as it was successfully used for grasp recognition by Palm and Iliev [2007]. The method is not tested for property 5. However, in Skoglund et al. [2008] a next-state-planner based on fuzzy time-modeling was introduced. Since next-state-planners are known to be robust to perturbations [Ijspeert et al., 2002, Hersch and Billard, 2008], we believe that this should be the case for our system as well. This will be tested in our future work.

Alternative method for encoding human motions are described in section 2.4.

3.3.2 Trajectory Segmentation

One simple and effective way to segment a manipulation demonstration is by measuring the mean squared velocity of the demonstrator's end-effector. This is a technique equivalent to the one developed by Fod et al. [2002], but instead of using the mean squared velocity of the joints angles we use the end effector's velocity. The mean squared velocity is given by:

$$MSV(t) = \sum_{i=1}^{3} \left(\frac{dx(i,t)}{dt} \right)^2 \tag{3.18}$$

where $i = 1...3$ is the number of the spatial coordinate, t is the time, $dx(i,t) \in \mathbb{R}^1$ is the position difference $dx(i,t) = x(i,t+dt) - x(i,t)$ between two samples, dt is the time difference between two samples. A constant threshold ν for MSV is set to define the start time t_r of a motion. If $MSV(t) > k \times \nu$, where k is a multiplication factor, then $t = t_r$. The start time t_r is checked for each time instant t after a motion has stopped. A similar procedure is used to find the stop of a motion.

3.3.3 Automatic Task Assembly

The scenario we consider is to teach an industrial robot equipped with a vacuum gripper, shown in figure 3.10, how to execute a pick-and-place task. The demonstration is done under the assumption that the teacher's index finger is associated with the suction cup of the gripper. During demonstration, the fingertip is tracked by a motion capturing device, shown in figure 3.11. Initially, the demonstrator moves from a starting point to the desired pick-point, P_{pick}. Then, the demonstrator moves along a certain path towards the desired place-point, P_{place} and finally, back to the starting position. The collected data consists of position coordinates P, used for the following purpose:

- Detect the pick- and place-positions, P_{pick} and P_{place}.

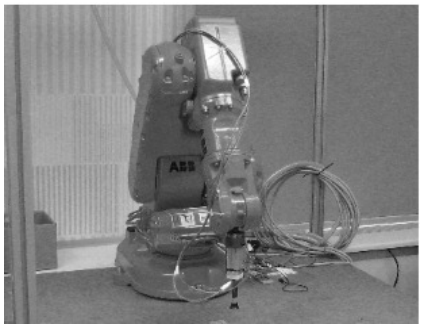


Figure 3.10: *The manipulator in our experimental setup, an ABB IRB140 manipulator equipped with a vacuum gripper.*

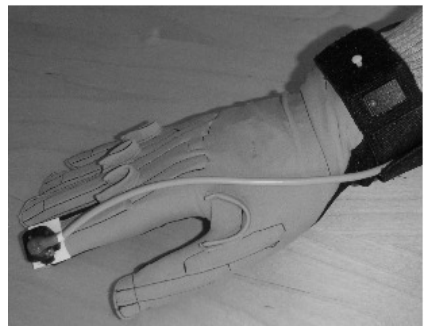


Figure 3.11: *The 6D-tracker, mounted on a data glove, is shown that was used to capture the human demonstration.*

- Reconstruct the desired trajectory that the robot should travel from P_{pick} to P_{place} (`FollowTaj`).

In summary, the steps from the demonstration to the compilation of instructions are:

1. A human demonstration is captured and transformed into robot's reference frame.
2. Trajectories are modeled using fuzzy modeling, to smooth and and reduce noise from the sensors.
3. Trajectories are segmented to extract the points where the motions start and end (see section 3.3.2).
4. Extracted motions are decomposed into skills as those described in section 3.3.4.
5. Each skill is automatically translated into robot-specific code.
6. The complete task is executed on the real ABB IRB140 manipulator.

For step 4 in the above list it is important to note that the task is known in advance, making it possible to describe the task as a sequence of skills. These skills are designed specifically for the task, and can be executed on most 6 DOF serial manipulators. The skills controlling the grasp and release action are specific to the type of gripper used.

In the scenario, the demonstrator and the robot share the same workspace. Our assumption is that the demonstrator knows the manipulator's structure such as workspace and possible motions, making a complicated data preprocessing unnecessary.

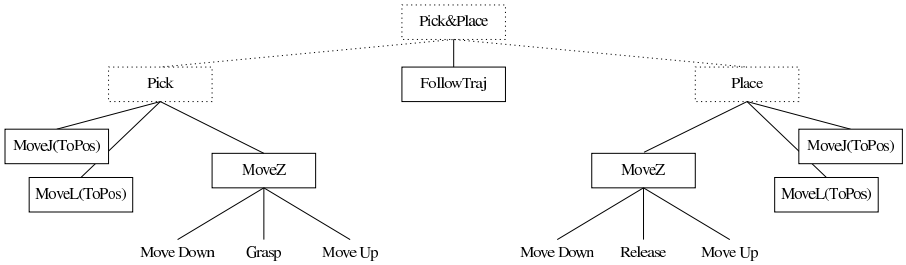


Figure 3.12: *Decomposition of a task into skills. The solid boxes are the implemented skills.*

3.3.4 Skills

To distinguish the different segments of a demonstrated motion, the captured data needs to be segmented. Here we follow the technique described in section 3.3.2. These segments are the basic skills, which are highly dependent on the task to perform. The decomposition of a Pick-and-Place task into skills is illustrated in figure 3.12. `MoveToPosJ` moves the manipulator's end-effector to the point where it can "hook on" to the demonstrated trajectory. `MoveToPosL` moves the manipulator's end-effector to the desired point linearly in the work space coordinates (for an introduction to robot motions, see Appendix A.4). `MoveZ` is a the skill for making a search motion when there is uncertainty of how to approach an object. `FollowTraj` takes a sequence of points, with relatively high granularity, as the input and executes the motion linearly between these points.

By using skills reflecting the commands available in the robot language of the manipulator, we achieve a simple implementation. The basic skills, which all other high level tasks (in this work) consists of, are:

- `MoveToPosL` which moves linearly (Cartesian space) to a predefined point,
- `MoveToPosJ` which moves to a predefined point,
- `MoveToPosZ` which moves in an up-down direction,
- `FollowTraj` which follow a demonstrated trajectory,
- `Grasp` which grasps an object,
- `Release` which releases an object,

The two first skills are usually implemented on standard industrial manipulators, yet needed as primitives for many tasks. In our setup the two skills controlling grasp and release are very simple due to the design of the vacuum gripper. However, for more complex grippers, or even anthropomorphic hands, a complex set of skills is needed [Palm and Iliev, 2006].

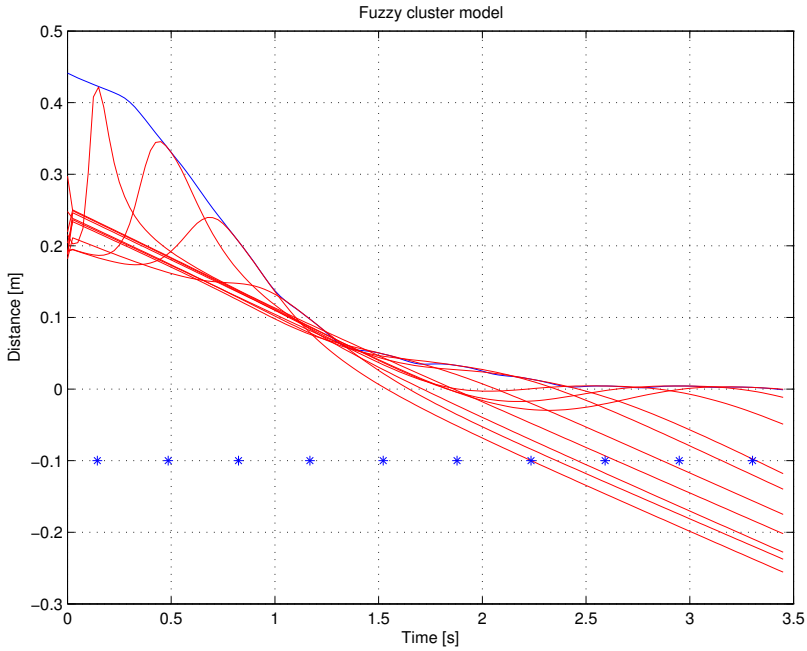


Figure 3.13: *Fuzzy clustering principle. The blue stars are the cluster centers positions, the lines are the output from each local linear model and the blue line is the total approximated model. The weights of each local model and the offsets are not shown.*

When the trajectory is transfered to the manipulator's controller, it is written in a compact meta format representing the local program of the controller processes. By doing so, we assume the lower control levels of the manipulator, such as inverse kinematics (see Appendix A.1), constraints of the workspace, generation of the trajectory with a higher granularity etc., to perform the required task in a proper way.

The skill `FollowTraj` is implemented using trajectory modeling based on fuzzy time-modeling described in section 3.1.3. Between the points detected as pick-and-place, the trajectory chosen by the demonstrator should be followed to reproduce the demonstration and to make the motion humanlike. The skill `FollowTraj` implements the trajectory following and uses a so-called time clustering method [Palm and Iliev, 2006] based on Takagi-Sugeno fuzzy modeling [Takagi and Sugeno, 1985] and fuzzy clustering [Palm and Stutz, 2003]. The basic idea is to consider the three end effector coordinates of the hand as model outputs and the time instants as model inputs, see figure 3.2. Figure 3.13 illustrates a TS fuzzy model.

Before the manipulator performs a grasp or release operation, the approach phase of the gripper towards the object is of high importance. In this section we only consider approach motions towards the object perpendicular to the

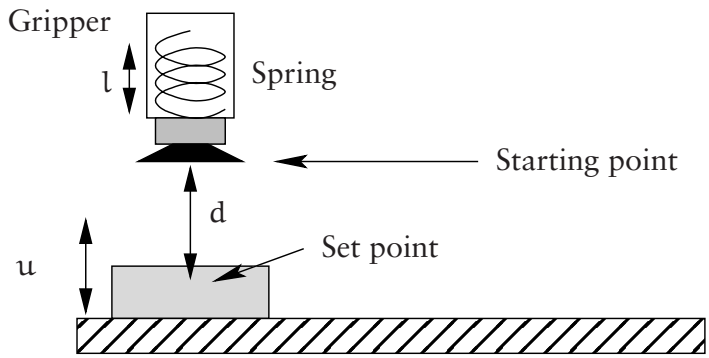


Figure 3.14: *The gripper and the spring.* d*, the distance target set point and the starting point, is dependent of the sensor inaccuracy,* u*. The length that the spring is compressed,* l*, is determined when the switch turns on, indicating a resistance.*

surface of the table (this also implies objects with flat top surfaces), which is a simplification derived from the design of the 1 DOF vacuum gripper. The position and orientation of the table are known since the manipulator is mounted on the table (see figure 3.10). However, the height of the object to be picked is unknown resulting from the uncertainty of the sensor, thus, the approach skill has to deal with this uncertainty. Therefore, the manipulator's gripper is equipped with a switch that detects when a spring is compressed to a certain extent (illustrated in figure 3.14). The downwards motion immediately stops when the switch detects contact, and the appropriate action (grasp or release) is performed.

A grasp operation is distinguished from a release operation by two discrete states, internally represented in the `MoveToPosZ` skill. When performing a grasp or release operation the manipulator is given a set coordinate to move in the direction of the object and search for contact with it. When the switch detects a certain resistance (that is, the spring is compressed a length, l) the motion stops. The distance d determines the starting point, derived from the inaccuracy of the sensor that performs the motion capturing, typically factor of 1.1 ~ 2.0. The approach skill is implemented using the `SearchL` command in RAPID (see the ABB specific programming languages, facilitating instructions for manipulator motions).

3.3.5 Experimental Results

The experimental setup consists of a 6D magnetic tracker, Polhemus Fastrak, and a 6 DOF industrial manipulator, the IRB140 from ABB Robotics. A vacuum gripper is mounted on the manipulator. The gripper has a magnetic switch and a spring. We have selected a pick-and-place task for our work since it is a

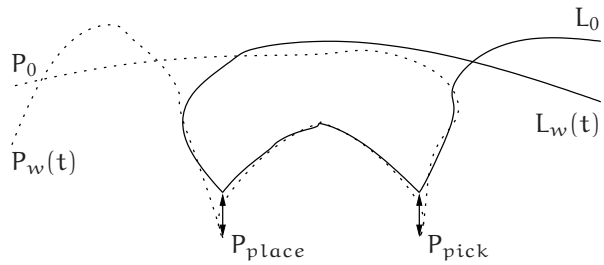


Figure 3.15: *An illustration of the trajectories, where the dotted line is the demonstration, and the solid line the manipulators end effector path. The manipulator's initial and final positions are denoted* L_0 *and* $L_w(t)$ *respectively, and the demonstrator's* P_0 *and* $P_w(t)$. t.

common task for manipulators. The demonstrator stands in front of the manipulator so that both share the same workspace. The demonstration consists of pointing at two different locations in the workspace and returning to the initial position. The first detected position is the pick position, denoted by p_{pick}, and the second detected position is the place position, denoted by p_{place} (see figure 3.15). The manipulator's initial and final positions are denoted by L_0 and $L_w(t)$ respectively in figure 3.15, and P_0 and $P_w(t)$ denotes the demonstrator's initial and final positions, and t indicates the time. The different positions at the start and end of the trajectory are resulting from the fact that demonstrator and manipulator are not in the same start and final position, but they share the same workspace and therefore the pick- and place points, and the trajectory in between are the same.

Figure 3.16 shows three example trajectories of the three different pick-and-place positions. Figure 3.17 shows three sample velocity profiles where the segmentation has detected three start of movements and their respective ends in each demonstration. The MSV threshold was set to 40 and the multiplication factor to 1.5 for this specific example. Through figure 3.18, 3.19 and 3.20, we can see that the models cover correctly the task trajectory, as performed by the human. The pick-and-place points as reflected by both the model and the raw data show a discrepancy of 3.62 mm, which is tolerable for this type of tasks to be performed by the robot arm.

The trajectories are segmented, and fuzzy time-modeling is performed between the pick to the place position. The result shows a good modeling quality, with a slight displacement in time.

The different illustrations in figure 3.16 to figure 3.20 show a series of three sets of measurement for the same task (pick-and-place), between two demonstrated points in the workspace. We can first establish that the data obtained by repetition of the task shows similar profiles, and the models extracted from these sets of data show both similarities to their respective sets of data.

The sampling rate of the tracker is approximately 12 Hz, for position and orientation data, in reference to a fixed point. When incorporating a data glove for capturing the demonstrator's hand motions the tracker will be mounted on the wrist, and the data glove provides the transformation to the end-effector. With a fixed coordinate system on both the manipulator and the demonstrator it is straight forward to transform the local coordinate system of the demonstrator into robot coordinates. While being accurate in office environments the magnetic tracker is less suitable if there are any large metal parts in the neighborhood of the probe. In our experiments, the position error could be over 40 mm for demonstrations near the robot. When we replicated the workspace of the robot in another area with no surrounding metal parts, the position error was reduced to under 4 mm. The reason for the degrading performance in the robot cell was the amount of metal in manipulator and frames of the cell, which distorts the magnetic field. A passive motion capturing device, such as an optical measuring device used by Skoglund et al. [2006], would remove this effect. A visual system could be another option, but most vision systems have lower accuracy and problems with occlusions, in comparison with the 6D-tracker.

A video of the task performed is available at the authors home page: http://www.aass.oru.se/Research/Learning/arsd.html.

3.3.6 Conclusions

In this section we have discussed a method for trajectory modeling and task reconstruction based on skills, linking high-level human instructions to particular robot/gripper functionalities. In our setup, the demonstrator is aware of the function of the skills. This awareness reduces the risk for misinterpretations and infeasible instructions to the robot. The pick-and-place task can be mapped to a sequence of skills that are scalable for different sizes of the work space of the manipulator. A wearable input device captures the motions of the demonstrator by recording the coordinates of pick-and-place positions and the path in between. We use the mean squared velocity to segment the recorded motions and extract the pick- and place positions. We apply a fuzzy time-modeling method to approximate the desired path in space. The method shows excellent modeling performance and preserves dynamical (humanlike) properties of the motion.

3.4 Summary

In this chapter we have presented two methods for PbD. The first one was an approach to model-free mapping from human arm position to robot position of the end-effector. In the second method, a task (pick-and-place) is described hierarchically as a set of skills where a demonstration is mapped onto these skills. To implement the skill which follows a demonstrated trajectory a method

called fuzzy time-modeling is introduced. The skill "grasp" although being implemented in a very simple manner because of the simple gripper, provides experience on how built-in autonomy and sensing–even on a basic level–can alleviate the inaccuracy problem related to motion capturing devices.

In all experiments in this chapter, the accuracy of the motion capturing device (the ShapeTape and the magnetic tracker) was a source of problems. In the first experiment this caused poor repeatability in the positioning of the end-effector, and in the second experiment it was hard to locate and manipulate objects. Therefore, in later performed experiments we replaced the ShapeTape and the 6D magnetic tracker by a new more accurate motion capturing system, described in section 4.3.1. To allow programming of more complex grippers, data from the demonstrator's hand configuration are also used in subsequent experiments, described in chapter 4.

The fuzzy time-modeling method shows good performance in modeling of human trajectories. Furthermore, it fulfills four of five desired properties for skill encoding, enlisted in section 3.3.1.

The next chapter goes into detail on some critical issues not addressed in this chapter, namely: how to map human arm and hand motions to a robot despite different morphologies; how to learn and generalize from multiple demonstrations.

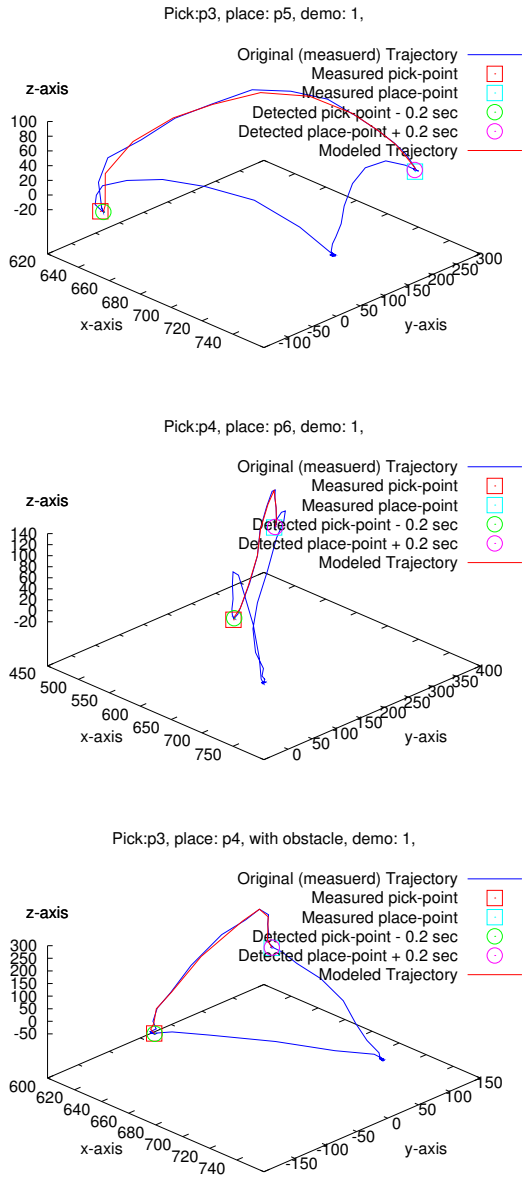


Figure 3.16: *Demonstrated trajectories (blue line) together with the modeled trajectories and the detected pick-and-place positions (squares) and the modeled pick-and-place positions (rings). Scales are in [mm].*

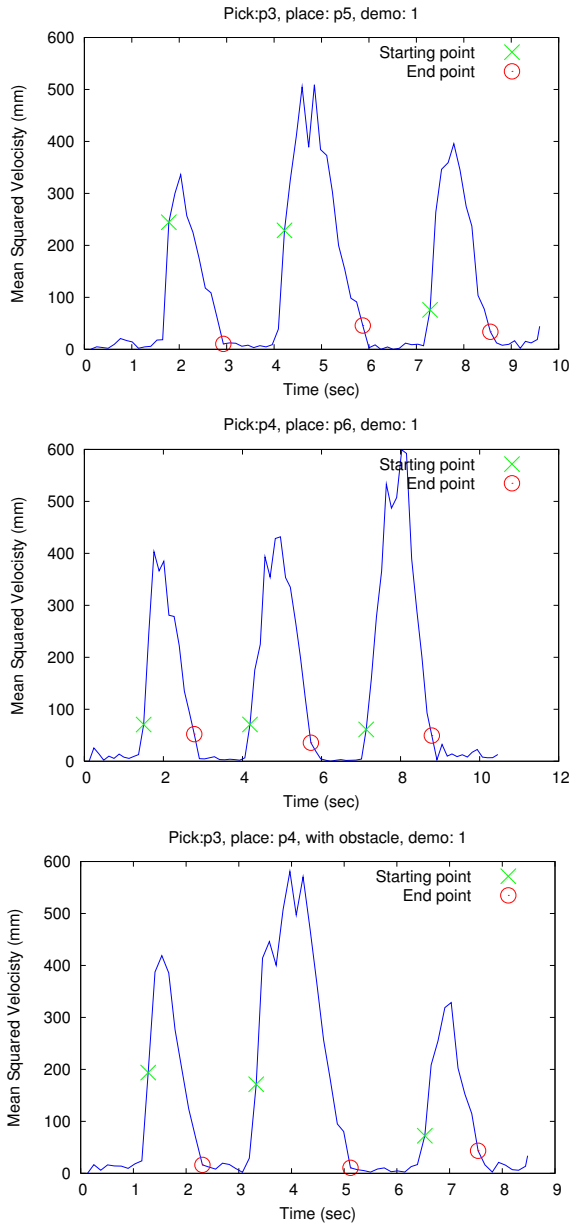


Figure 3.17: *Velocity profiles of the demonstrations. Ring and cross marks where the segment process indicates the start and end of a motion.*

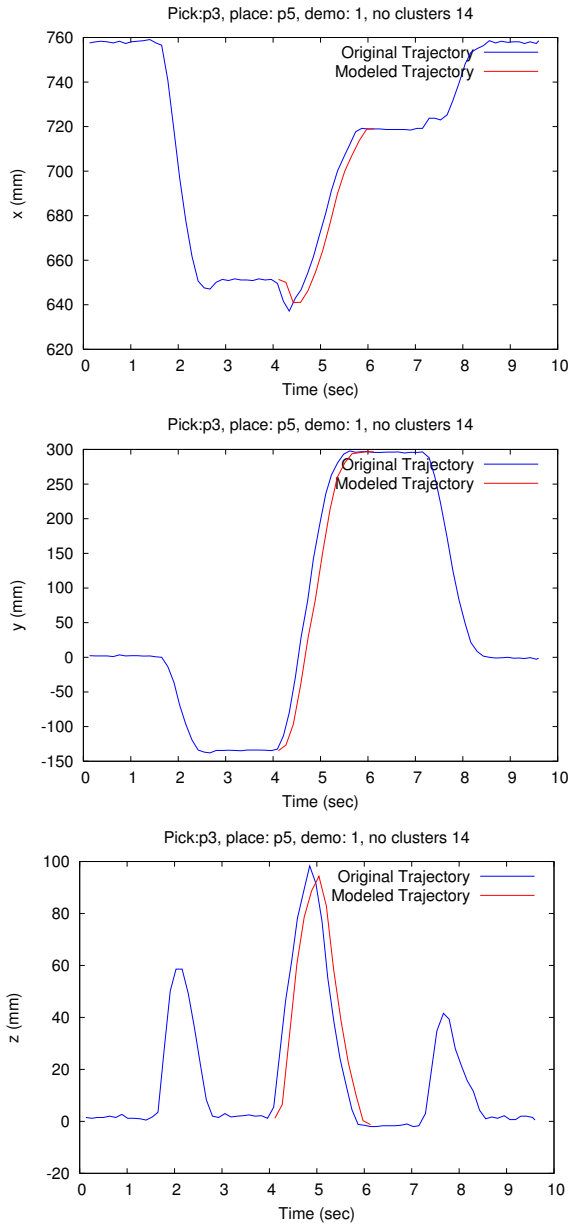


Figure 3.18: *Trajectory profiles obtained throughout measurement and task-model as function of time for task* P_3 *to* P_5.

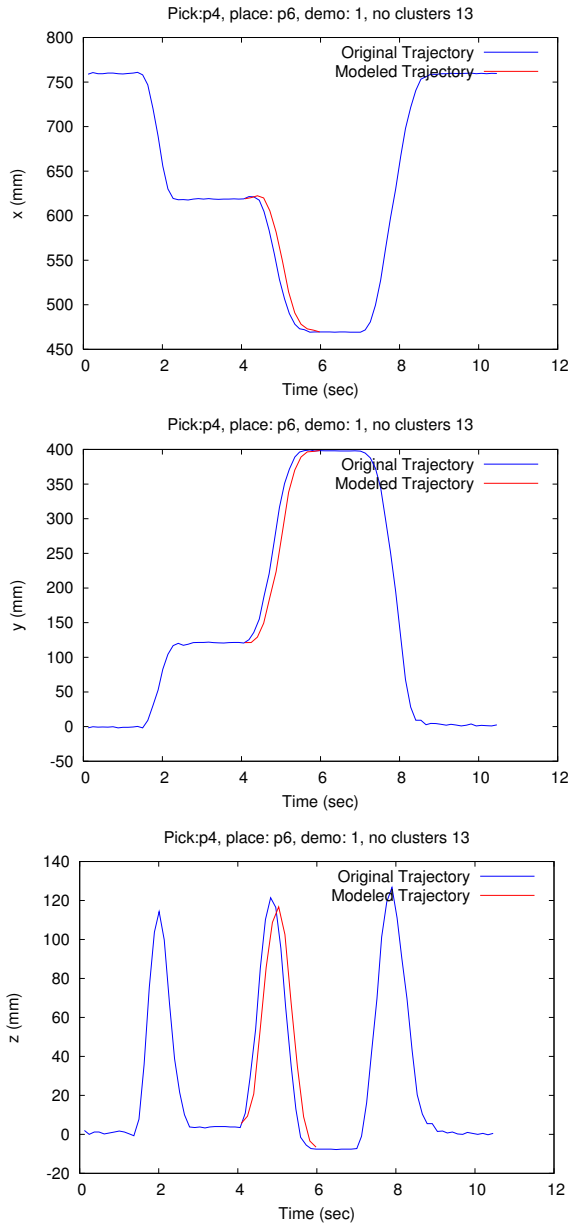


Figure 3.19: *Trajectory profiles obtained throughout measurement and task-model as function of time for task* P_4 *to* P_6.

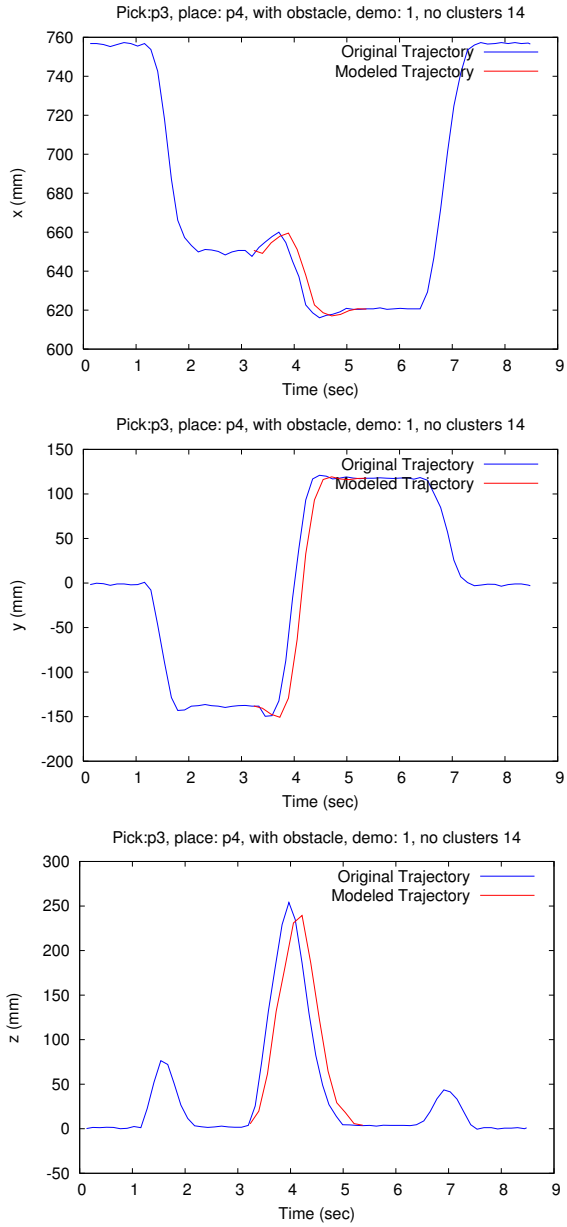


Figure 3.20: *Trajectory profiles obtained throughout measurement and task-model as function of time for task* P_3 *to* P_4.

Chapter 4
Trajectory Generation in Hand-State Space

This chapter presents an approach to reproduction of human demonstrations in a reach-to-grasp context. The demonstration is represented in hand state space, a hand-object relative frame. By using the distance to the target object as a scheduling variable the robot's approach motion to the object is controlled. A trajectory generator is employed to plan the motion and is formulated as a next-state-planner. The planner produces an action from the current state instead of planning the whole trajectory in advance, and can thereby compensate for differences in modeled and actual trajectory.

One major problem, discussed earlier in section 2.1, is the different morphology between human and robot, known as the correspondence problem in imitation [Nehaniv and Dautenhahn, 2002]. The behavior matching needs to be addressed to reproduce an action. That is, what action can an imitator apply that corresponds to the demonstrated action. The next-state-planner will be shown to provide this mapping since it produces a *robot version* of the demonstrated behavior. Another problem is the difference between the initial locations of the human demonstrator and the robot while aiming for the same target object. When executing the demonstrated motion, different initial locations might force the robot into unreachable parts of the workspace or singular arm configurations even if the demonstration is perfectly feasible from the human viewpoint. The duration of the robot motion will also differ from the demonstrated motion resulting from different initial positions. The next-state-planner generates trajectories from the robot perspective and exploits the constraints imposed by multiple demonstrations to adapt the trajectory to the morphology of the robot. That is, it allows the robot to reproduce the task in its own way.

In this chapter we focus on goal-directed imitation of reaching and grasping, where the essential properties of the target object and the desired motion

profile are obtained from demonstrations. The suggested approach can be outlined as follows. Firstly, a demonstration is recorded, segmented and converted into a hand-object relation trajectory, the so called *hand-state space*. The robot can then extract the essential grasp related properties of the target object (its affordances). Secondly, the demonstrated task is reproduced using a trajectory generator that accounts for experience and the current robot configuration.

In chapter 3 it was shown how a pick-and-place-place task was transformed from a human demonstration into desired trajectories for the robot. However, the grasping part; i.e., how to pick up an object, was implemented in a simple manner to match the limited capabilities of the robot gripper. Moreover, the trajectory generation used in chapter 3 was set to follow the demonstrated trajectory at a predetermined distance to the target. This strategy does not address the problem of morphological difference between demonstrator and imitator. To solve the problem of morphological differences this chapter presents a trajectory planner that includes statistical knowledge from multiple demonstrations.

The approach presented in this chapter has four goals. First, to interpret the demonstrated task and reproduce it in a way feasible for robots, taking the morphological differences into account. Second, the robot shall generalize human demonstrations to use an earlier learned skill for new similar unseen scenarios. Third, it shall coordinate reaching and grasping motions into a coherent goal-directed motion. Fourth, the robot shall be enabled to learn from its own experience and improve grasp performance over time.

To illustrate the approach, two series of experiments will be presented using slightly different planners. The first set of experiments is done using a simulated robot, originally presented by Skoglund et al. [2008]:

- The first experiment shows how demonstrated trajectories can be reproduced and how the correspondence problem associated with different initial positions and morphological differences can be solved.

- The second experiment illustrates how the robot generalizes its knowledge to reproduce the demonstration when the object is placed at random positions in the workspace.

The second series of experiments includes tests with a real industrial manipulator. These experiments extend the approach to grasping, specifically addressing the position accuracy to enable grasping, presented by Skoglund et al. [2009]. Four experiments describe how human demonstration of goal-directed reach-to-grasp motions can be reproduced by a robot. Specifically, the generation of reaching and grasping motions in pick-and-place tasks is addressed.

- The first experiment is a simulation of an autonomous grasp performed from different poses in relation to the target. This will show how accurate the positioning of the end-effector needs to be to execute a successful grasp. It is important to know when the end-effector is in a position where the grasp execution can be started.

- The second experiment replicate the first set of experiments by generating trajectories to execute on a real robot and discuss problems associated with real manipulation.
- The third experiment illustrates how the robot generalizes its knowledge for new positions of the object. It reproduces the demonstration regardless of the initial position of the robot and the position of the object. The goal of this experiment is to investigate how well each model can generalize across the workspace. This is related to the number of models needed for the robot to perform a successful reaching-to-grasp action; good generalization ability means that fewer models are needed.
- The fourth experiment is done to assess the reaching and grasping as an integrated process. A complete pick-and-place task is demonstrated and executed by the robot.

The contributions of the work presented in this chapter are as follows:

1. We introduce a novel approach using a *next-state-planner* based on the *fuzzy modeling* approach to encode human and robot trajectories.
2. We apply the *hand-state concept* [Oztop and Arbib, 2002] to encode motions in hand-state trajectories and apply this in PbD. The hand-state description is the link between human and robot motions.
3. The combination of the next-state-planner and the hand-state approach provides a tool to address the *correspondence problem* resulting from the different morphology of the human and the robot. The experiments show how the robot can generalize and use the demonstration despite its fundamentally different morphology.

One advantage of this approach over trajectory averaging, see for example [Delson and West, 1996] or [Calinon et al., 2007], is that one of the human demonstrations is used instead of an average which might contain two essentially different trajectories [Aleotti and Caselli, 2006]. By capturing a human demonstrating the task, the synchronization between the reach and the grasp is also captured, which we demonstrate in the experiment in section 4.2. Other ways of capturing the human demonstration, such as kinesthetics, cannot easily capture this synchronization.

This chapter is organized as follows: In section 4.1 we review the hand-state hypothesis and related work. Furthermore, we describe methods for analyzing the demonstration. In section 4.2 the trajectory generation based on the hand-state approach is presented. A different version of the trajectory generation is presented in section 4.3 where several additional experiments are presented. Section 4.4 summaries this chapter.

4.1 Interpretation of Human Demonstrations in Hand-State Space

To create the associations between human and robot reaching/grasping we employ the hand-state hypothesis from the Mirror Neuron System (MNS) model of Oztop and Arbib [2002]. The aim is to mimic the functionality of the MNS to enable a robot to interpret human goal-directed motions in the same way as its own motions. Following the ideas behind the MNS-model, both human and robot motions are represented in hand-state space. A hand-state trajectory is defined as the evolution of the pose and configuration of the hand in relation to the target object. That is, it encodes the goal-directed motion of the hand during reach and grasp. It will be shown that the hand-state space serves as a common language for the demonstrator and the robot and preserves the necessary execution information. Hence, a particular demonstration can be converted into executable robot code and experience from multiple demonstrations is used to control/improve the execution of new tasks. Thus, when the robot tries to imitate an observed reach and grasp motion, it has to move its own hand so that it follows a hand-state trajectory similar to the demonstrated one.

The approach is based on the assumption that human demonstrations contain motions associated with a particular task that can be interpreted, for example, pick-and-place. Grasp recognition can provide grasp type G_i, for example, a cylinder grasp or a precision grasp, where each grasp type corresponds to a motion primitive M_i which executes the associated type of grasp. A grasp type is also associated with object affordance A_i, which means that the object affords a type of grasp. For example, a cup affords a cylinder grasp around its body. The position of the object can be retrieved by a vision system, or it can be estimated from the grasp type and the hand pose. If hand motions with respect to a potential target object are associated with a particular grasp type G_i, it is assumed that there must be a target object that affords the observed grasp type, with the affordances A_i. The set of affordances for different grasps G_i is defined *a priori* or learned from a set of training data. In this way, the robot is able to associate observed grasp types G_i with their respective affordances A_i. Once the target object is known, the hand-state can also be defined. According to Oztop and Arbib [2002], the hand-state must contain components describing both the hand configuration and its spatial relation with respect to the affordances of the target object. Thus, the hand-state is defined in the form:

$$H = \{h_1, h_2, \ldots h_{k-1}, h_k, \ldots h_p\} \tag{4.1}$$

where $h_1 \ldots h_{k-1}$ are *hand-specific components* which describe the motion of the hand during grasping. The remaining components $h_k \ldots h_p$ describe the motion of the hand in relation to the object. Thus, a hand state trajectory contains a record of both the reaching and the grasping motions as well as their synchronization in time and space. Note that the hand-state components are

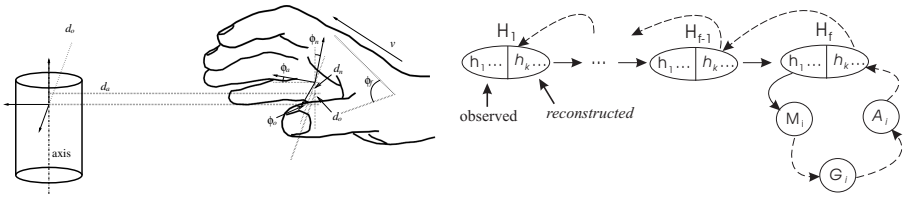


Figure 4.1: Left: *The hand-state describes the relation between the hand pose and the object affordances.* **Right:** *Reconstruction of hand-state trajectory.*

specific to the kinematic structure of the hand and the definition of the object affordances. This means that some hand-state components will be defined differently for different grasp types since they involve different object affordances. If a robot should imitate a human, we have to define H in such way that it matches the capabilities of particular types of end-effectors, e.g., dexterous robot hands, two-finger grippers, or even the human hand.

In the PbD framework, $h_1, \ldots h_p$ must be such that they can be recovered from both human demonstrations and the perception system of the robot. That is, the definition of H is *perception invariant* and can be updated from arbitrary type of sensory information. To the left in figure 4.1 the definition of the hand state used in this chapter is shown.

Let the human hand be at some initial state H_1. Then the hand moves along a certain path and reaches the final state H_f where the target object is held by the hand [Iliev et al., 2007]. That is, the recorded motion trajectory can be seen as a sequence of states, i.e.,

$$H(t) : H_1(t_1) \rightarrow H_2(t_2) \rightarrow \ldots \rightarrow H_f(t_f) \tag{4.2}$$

When a complete recording of the demonstration becomes available, the robot can recognize the grasp type from M_i. This can be done for example by using the grasp recognition method introduced by Palm and Iliev [2007]. Then, applying the a priori defined associations between grasp types, affordances and motion primitives M_i the unknown hand-state components can be reconstructed. This process is done in a few steps, see right of figure 4.1:

1. Define the type of affordance A_i corresponding to observed grasp type G_i by using the $G_i \rightarrow A_i$ associations.

2. Substitute the reconstructed A_i in components $h_k - h_p$ to obtain the complete hand-state at H_f.

3. Update the remaining states of the hand-state trajectory from the final state H_f to the initial state H_1. Encode the trajectory using fuzzy time-modeling.

To determine the hand-state representation of a demonstration the robot needs to have access to the complete motion trajectories of the teacher's hand since the motion must be in relation to the target object. This means the hand-state trajectories only can be computed *during* a motion if the target object is known in advance.

Since the human demonstration $H_d(t)$ cannot be executed by the robot without modification in the general case, we have to construct the *robotic* version of $H_d(t)$, denoted by $H^r(t)$, illustrated to the left in figure 4.2. To find $H^r(t)$ a mapping from the human grasp to the robot grasp is needed, denoted T_h^r. This mapping is created as follows. We can measure the pose of the demonstrator hand and the robot hand holding the same object at fixed position and obtain T_h^r as a static mapping between the two poses. The pose of the robot hand at the start of a motion defines the initial state H_1^r. The target state H_f^r will be derived from the demonstration by mapping the goal configuration of the human hand H_f into a goal configuration for the robot hand H_f^r, to the left in figure 4.2, using the transformation T_h^r:

$$H_f^r = T_h^r H_f \tag{4.3}$$

where T_h^r is the transformation from human to robot. For the power grasp the robot hand is positioned so the grasp is expected to be successful at H_f^r. Next the human hand position H_f in hand state space and the robot hand position H_f are use to compute the transformation T_h^r from human to robot obtained by:

$$T_h^r = H_f^r H_f^{-1} \tag{4.4}$$

It should be noted that this method is only suitable for power grasps. In the general case it might produce ambiguous results or rather inaccuarate mappings. For a predefined grasp type T_h^r becomes known immediately after we have access to H_f.

4.2 Next-State-Planner

In this section we will present the fuzzy modeling based next-state-planner, which generates the robot trajectory. Section 4.2.1 describes the modeling which is needed by the planner. Let us first see how the hand-state is implemented for trajectory generation.

4.2.1 Trajectory Modeling

The encoded trajectory $H_d(t)$ is then used for generating the robot's reaching trajectory towards the target object. To provide the robot with more knowledge about the task, multiple demonstrations are recorded, thus imposing trajectory constraints. Finally, the trajectory generator produces a trajectory from H_1^r to

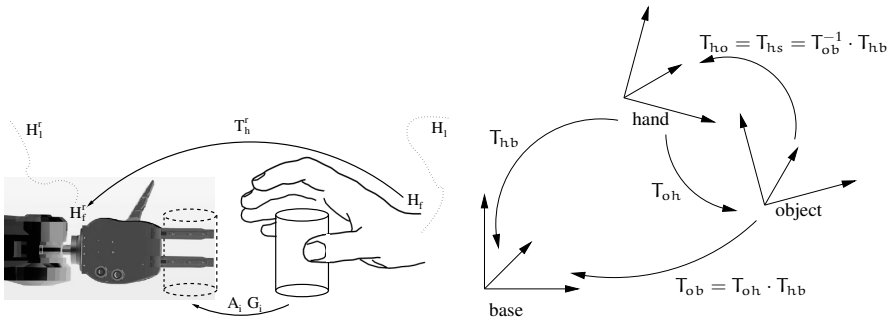


Figure 4.2: Left: *Mapping from human hand to robotic gripper.* **Right:** *The transformation between different frames.* T_{ee} *is the end-effector transformation,* T_{ob} *is the object transformation and* T_{hs} *is the transformation from object to hand, which describes the object-hand part of the hand-state.*

H_f^r using one demonstration, but with the constraints from several demonstrations. $H_d(t)$ is also used for scheduling the activation of the proper grasping motion primitive.

Having the initial and the target states defined, we have to generate the trajectory between the two states. In principle, we could transform $H_d(t)$ using equation 4.3 in such way that it has its final state in H_f^r. Then, the robot starts at H_1^r, approaches the displaced demonstrated trajectory and tracks it until the target state is reached. However, such an approach would not take trajectory constraints into account. Thus, it is also necessary to specify exactly how to approach $H_d(t)$ and what segments must be tracked accurately. Moreover, $H_r(t)$ has to synchronize the reaching motion driving the arm with the grasp primitive driving the hand.

The workspace restrictions of the robot also have to be considered when creating trajectories. A trajectory might contain regions which are out of reach, or two connected points on the trajectory require different joint space solutions (see Appendix A.4), thus, the robot cannot execute the trajectory. To avoid or remedy the effect from this problem the manipulator must be placed at a position/orientation with good reachability. Other solutions include a mobile platform, larger robot, or more degrees of freedom (DOF) to mimic the redundancy of the human arm.

Next, a hand-state trajectory must be constructed from the demonstrated trajectory. From the recorded demonstration we reconstruct the end-effector trajectory, represented by a time dependent homogeneous matrix $T_{ee}(t)$. Each element is represented by the matrix:

$$T_{ee} = \begin{pmatrix} N_{ee} & O_{ee} & A_{ee} & Q_{ee} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.5}$$

where N_{ee}, O_{ee} and A_{ee} are the normal vector, the side vector, and the approach vector respectively. The last vector Q_{ee} is the position. The matrix T_{ee}

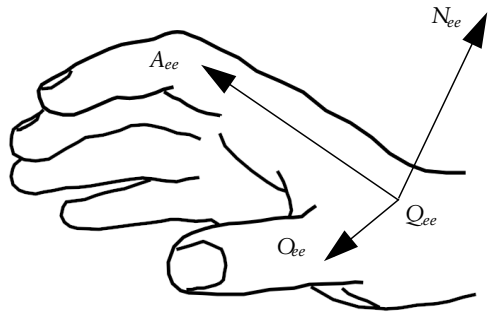


Figure 4.3: *Vectors definition in a human hand.* N_{ee} *is the the normal vector,* O_{ee} *the side (orthogonal) vector and* A_{ee} *is the approach vector. The vector* Q_{ee} *is the position of the point.*

is defined differently for different end-effectors, for example, the human hand is defined as in figure 4.3.

Given the applied grasp type, we can also estimate affordances of the grasped object. For example, when a cylindrical grasp is recognized, we estimate the affordances associated with a cylinder. These are: center position $P = [P_x\ P_y\ P_z]^T$, radius r and cylinder axis $R = [R_x\ R_y\ R_z]^T$. The position P is determined from $T_{ee}(t_f)$, where t_f is the time at the final position. The cylinder axis is determined by the orientation vector O_{ee} of T_{ee}. For a cylindrical object grasped with a power grasp the approach vector A_{ee} is free to rotate around the cylinder axis of the object. Therefore, the approach vector A_{ee} is set in the direction pointing from the base of the robot to the object. For grasping a plannar object both N_{ee} and O_{ee} are fixed, but the signs (positive or negative directions) are free to move so that the robot can choose the most convenient one of the two possible approach vectors A_{ee}.

Now, recall the definition of the hand-state from equation 4.1, we describe the object-hand components by a transformation matrix called T_{hs}. T_{hs} is represented in the object frame T_{ob}, which is $T_{ee}(t_f)$. The transformation to the end-effector frame relative to the object frame is expressed by:

$$T_{hs} = T_{ob}^{-1} \cdot T_{ee} \tag{4.6}$$

where T_{hs} is the hand-state transformation, T_{ee} is the hand frame and T_{ob} is the object frame. These transformations are illustrated in figure 4.2.

The hand-state representation equation 4.1 is invariant with respect to the actual location and orientation of the target object. Thus, demonstrations of object-reaching motions at different locations and initial conditions can be represented in a common domain. This is both the strength and weakness of the hand-state approach. Since the hand-state space has its origin in the goal object, a displacement of the object will not affect the hand-state trajectory. When

an object is firmly grasped then, the hand-state is fixed and will not capture a change in the object position relative to the base coordinate system. This implies that for object handling and manipulation the use of hand-state trajectories is limited.

Once the hand-state trajectory of the demonstrator is determined it has to be modeled, for reasons enumerated in section 3.3.1.

There is evidence that the internal models of arm dynamics found in biological systems are state-dependent rather than time-dependent [Conditt and Mussa-Ivaldi, 1999]. Therefore, when we transform human demonstrations into robot motions; we define *distance to object* d, as an additional scheduling variable for hand-state trajectories. However, to preserve the velocity profile from the human demonstration the distance to the target is modeled as a function of time using fuzzy time-modeling, see section 3.1.3. The inputs to the fuzzy modeling are the hand-state components and the Euclidean distance at each instance t of time.

$$d_E = \sqrt{(Q_{ee} - P)^2} \tag{4.7}$$

where Q_{ee} and P are the end-effector position and object position respectively.

The same procedure is applied to the hand-state trajectories. Two types of models are needed: one describing the distance to the object as a function of time; one modeling of the hand-state as a function of distance. In this section a general formulation of the hand-state is adopted to suite industrial grippers with two states: open and close. We formulate the hand-state as:

$$H(t) = [\phi_f(t)\ d_n(t)\ d_o(t)\ d_a(t)\ \phi_n(t)\ \phi_o(t)\ \phi_a(t)] \tag{4.8}$$

The first component is the only hand-specific one, describing the angle between the thumb and the index finger. The next three components, $d_n(t)$, $d_o(t)$ and $d_a(t)$, describe the distance from the object to the hand along the three axes n, o and a with the object as the base frame. The next three components, $\phi_n(t)$, $\phi_o(t)$ and $\phi_a(t)$, describe the rotation of the hand in relation to the object around the three axes n, o and a. The notion of the hand-state used in this section is illustrated in figure 4.1. The individual components denote the position and orientation of the end-effector as well as the opening and closing of the gripper, reflecting the limited capabilities of a parallel gripper.

The components of the hand-state, as a function of distance, are given by:

$$H(d) = [\phi_f(d)\ d_n(d)\ d_o(d)\ d_a(d)\ \phi_n(d)\ \phi_o(d)\ \phi_a(d)] \tag{4.9}$$

where the hand-state components are the same as in equation 4.8, and $d \in \mathbb{R}^1$ The role of the scheduling variable d is important since it expresses *when* the robot should move to the next state. However, the hand-state variables reflect *where* the hand should move. Thus, d synchronizes *when* (dynamics and synchronization) and *where* (desired path).

Note that with this simplified definition of H we cannot determine the actual human grasp type. This reflects the fact that the grippers we use in our experiments only are capable of one type of grasp. However, the method can be applied in a straightforward way also to complex robotic hands being capable of executing several grasp types, see [Iliev et al., 2007] for details.

The configuration of the human hand is used for grasp classification, described in work by Palm and Iliev [2006, 2007], where the grasps are classified according to the taxonomy by Iberall [1997]. The grasp type G, and it's associated set of affordances A are used to provide the frame in which the hand-state is described. Grasp classification is out of scope of this thesis, and only cylindrical or spherical grasps are used our experiments. Thus, the grasp type is assumed to be known $G = \{cylindrical, spherical\}$; the affordances are: position, size, and cylinder axis $A = \{width, axis\}$.

The Takagi-Sugeno fuzzy models are constructed from captured data from the end effector trajectory described by the nonlinear function:

$$\mathbf{x}(y) = \mathbf{f}(y) \tag{4.10}$$

where $\mathbf{x}(y) \in \mathbb{R}^n, \mathbf{f} \in \mathbb{R}^1$, and $y \in \mathbb{R}^m$. The parameter y can be the time t or the distance d. The modeling follows the same procedure as described in section 3.1.3, using equation 3.14 to 3.17.

From demonstrations, we obtain models of motions as in the form of equation 4.9. However, the robot's motion time MT_{robot} will differ from the demonstration since the initial conditions are different. MT_{robot} is computed at $t = 0$ before the robot starts executing the motion. Therefore, the model of the human motion needs to be adapted to fit the robot version. To adapt the model to the robots initial condition, we need to perform four steps:

1. Normalize the cluster centers positions C_i, where each cluster center is a point in time. Adjust the offsets $\mathbf{a}_i$ $(i = 1 \dots c)$ with respect to the position of the maximum cluster center c_{max}, by:

$$\begin{aligned} \tilde{C}_i &= C_i / C_{max} \\ \tilde{\mathbf{a}}_i &= \mathbf{a}_i / c_{max} \end{aligned} \tag{4.11}$$

2. Compute the new cluster centers positions C^{robot}, in time, and the new offsets a^{robot} by multiplying them by MT_{robot}, that is:

$$\begin{aligned} C_i^{robot} &= MT_{robot} * \tilde{C}_i \\ \mathbf{a}_i^{robot} &= MT_{robot} * \tilde{\mathbf{a}}_i \end{aligned} \tag{4.12}$$

3. Compute a scaling factor k that relates the demonstrated time duration to the robot time duration MT_{robot}, by:

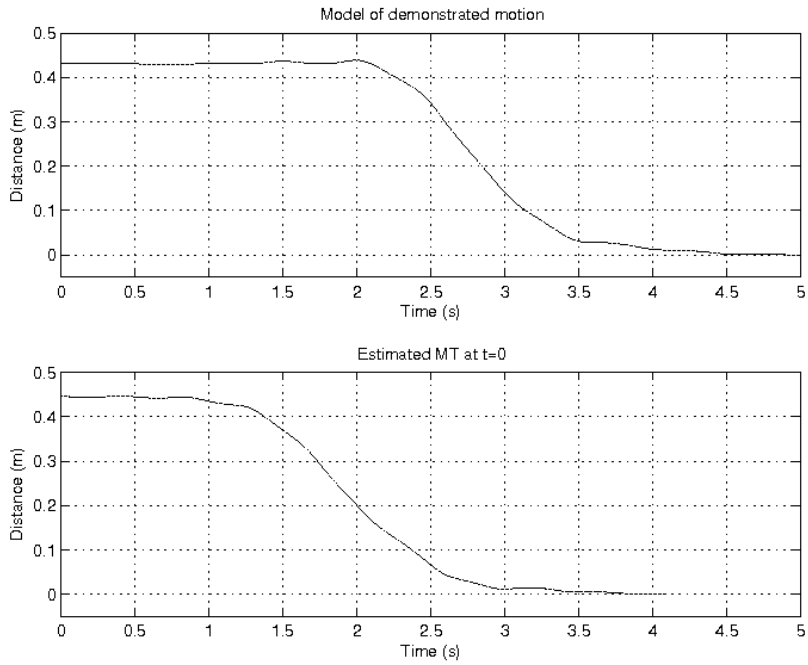


Figure 4.4: *The top graph shows the original model of the distance to the target as the time evolves. The bottom graph shows how the model is adapted to the robots domain. This estimate is made at the initial position, and might change during the execution.*

$$k = \frac{c_{max}}{MT_{robot}} \tag{4.13}$$

4. Finally, the distance at time t is given by:

$$d^{robot} = k \cdot \sum_{i=1}^{c} w_i(t) \cdot (\mathbf{A}_i \cdot t + \mathbf{a}_i^{robot}) \tag{4.14}$$

where d^{robot} is the robot's distance as a function of time, c is the number of cluster centers, and $\mathbf{a}^{new}$ are the robot's offsets. Note that the number of cluster centers is the same, it is only the position that has changed.

The demonstrated distance model, top graph in figure 4.4, is scaled using the above algorithm to a new duration shown in figure 4.4, bottom.

Since multiple demonstrations of a task is available to the robot we can compute statistics such as the variance of the motion. We exploit the fact that when humans grasp the same object several times they seem to repeat the same

grasp type which leads to similar approach motions. Based on that, multiple demonstrations of grasp type G using affordances A become more and more similar to each other the closer one gets to the target state. This implies that successful grasping requires an accurate positioning of the hand in an area near the object while the path towards this area is subject to less restrictions. Therefore, by looking at the variance of several demonstrations the importance of each hand-state component can be determined. The idea is that if the variance in certain components is low the trajectory should be followed, in contrast, if the variance is high the goal state should be approached. The variance of the hand-state as a function of the distance to target d is given by:

$$var({}^{k}h(d)) = \frac{1}{n-1}\sum_{i=1}^{n}({}^{k}h_i(d) - mean({}^{k}h(d)))^2 \tag{4.15}$$

where d is the Euclidean distance to the target, ${}^{k}h_i$ is the k^{th} hand-state parameter of i^{th} demonstration (from equation 4.8) and n is the number of demonstrations. Figure 4.5 shows how the variance decreases as the distance to the object decreases. This means that the position and orientation of the hand are less relevant when the distance to the target increases.

To preserve some human motion characteristics the robot can be made to move according to Fitts' law. Recall that Fitts' law from equation 2.1, in section 2.2.2 describes the tradeoff between speed and accuracy of reaching motions:

$$MT = a + b\log_2\left(\frac{2A}{W}\right)$$

where MT is the duration of the motion, A is the amplitude of the motion, equal to distance to the target at the start of the motion $A = d(t_0)$, W is the width of the object and a and b are coefficients.

Suppose that MT, $d(t_0)$ and W are known from demonstrations. Then we can solve for the coefficients a and b. When a robot is instructed to imitate the human demonstration, we cannot assume the amplitude of the motion to be the same as in the demonstration. Thus, we need to compute the desired duration of the robot's motion MT_{robot} if we want to preserve the characteristics of the human motion. Since we know the width of the object, the initial distance to the object and the coefficients a and b, we can apply Fitts' law to compute the time MT_{robot}.

4.2.2 The Goal- and Trajectory-Following-Planner

In this section we present a planner that balances its actions between *following a demonstrated trajectory* and *approach a target*, first presented by Skoglund et al. [2008]. Section 4.3 presents a different version of the same planner with

simplified dynamics and more experimental results, presented in [Skoglund et al., 2009].

To generate the robot trajectory a next-state-planner is used, which is inspired by the Vector Integration To Endpoint (VITE) planner (discussed in 2.7) suggested by Bullock and Grossberg [1989]. The VITE planner is a biologically inspired planner for human control of reaching motions. A next-state-planner plans one step ahead from its current state. This contrasts to traditional approaches that plan the entire trajectory in advance. The next-state-planner approach requires a control policy, a set of equations describing the next action from the current state and some desired behavior.

The proposed next-state-planner generates a hand-state trajectory for the robot using the TS fuzzy-model of a demonstration. It also includes a weighting factor obtained from multiple demonstrations, based on the variance in hand-state space.

One of the first researchers to use the next-state-planner approach in imitation learning were Ijspeert et al. [2002]. They encode the trajectory in an autonomous dynamical system with internal dynamic variables that shapes a "landscape" used for both point attractors and limit cycle attractors. Their approach shows robustness to perturbations and was demonstrated on a humanoid robot by performing gesturers and a tennis swing. However, object manipulation was not address in their work.

For controlling a humanoid's reaching motion Hersch and Billard [2008] considered a combined controller with two VITE controllers running in parallel. One controller acts in joint space and while the other one acts in Cartesian space. Separately, the controller acting in Cartesian space produces straight motions to the target. The controller which acts in joint space, if used separately, produces a trajectory that avoids joint limits and thereby singular configurations. By combining the two controllers, a coherent trajectory is produced that takes constraints from both domains into account. In a PbD application the approach suggested by Hersch and Billard [2008] can remedy the effect of unreachable joint configurations if the planning is done in Cartesian space. However, this might violate the constraints imposed by the demonstration.

To generate reaching motions and avoiding obstacles simultaneously Iossifidis and Schöner [2006] used attractor dynamics. The target object acts as a point attractor on the end effctor. The end-effector as well as a redundant elbow joint avoids an obstacle as the arm reaches for an object. They show how an obstacle avoiding behavior can be included in a next-state-planning approach. The end-effector is assumed to move in the direction of the approach vector (see figure 4.3 for a definition using a human hand). In PbD this is not the general case, the hand might move in some arbitrary direction.

In our approach, a human demonstration is used to guide the robot to grasp an object. Our use of a dynamical system differs from previous work, i.e., [Hersch and Billard, 2006, Ijspeert et al., 2002, Iossifidis and Schöner, 2004], in the way of how to combine the demonstrated path with the robots own plan. The

use of hand-state trajectories distinguishes our work from most previous work on imitation. Most approaches in the literature use the joint space for motion planning while some other approaches use the Cartesian space, according to Ijspeert et al. [2002].

The proposed next-state-planner generates a hand-state trajectory for the robot using the TS fuzzy-model of a demonstration. As the resulting $H^r(t)$ is formulated in Cartesian space the inverse kinematic provided by the controller for the robot arm is used. The TS fuzzy-model serves as a skill for controlling the arm's reaching motion. The initial hand-state of the robot is determined from its current configuration and the position and orientation of the target object since these are known at the end of the demonstration. Then, the desired hand-state H_d^r is computed from the fuzzy time-model (equation 3.16). Then the desired hand-state H_d is fed to a hand-state trajectory generator. Figure 4.7 shows the architecture of the next-state-planner. The planner works as follows: first, the distance to the target is computed $d(t_0) = A$. Then the duration of motion MT is computed using Fitts' law (see equation 2.1). Since the position and affordances of the object are assumed to be known the initial hand-state of the robot can be computed for initialization. Then, the desired distance d_d is obtained from the TS fuzzy-model (equation 4.14), and the desired hand state H_d from the TS fuzzy-model (equation 3.16). The planner has the following dynamics:

$$\ddot{H} = \alpha(-\dot{H} + \beta(H_g - H) + \gamma(H_d(d) - H)) \tag{4.16}$$

where H_g is the hand-state goal, $H_d(d)$ the desired state at distance d, H is the current hand-state, $\dot{H}$ and $\ddot{H}$ are the velocity and acceleration respectively. α is a positive constant and β, γ are positive weights for the goal and tracking point, respectively. The goal hand-state H_g is obtained from the fuzzy clustering model at the modeled final time step. The desired hand-state value $H_d(d)$ at distance d is computed using the desired distance at the current time step and uses the fuzzy model at that distance, equation 3.16 rewritten using the applicable terms:

$$H(d) = \sum_{i=1}^{c} w_i(d) \cdot (\mathbf{A}_i \cdot d + \mathbf{a}_i) \tag{4.17}$$

By looking at the variance in the hand-state we can conclude that as the distance to the target decreases the variance in hand-state also decreases, see figure 4.5. Hence, the closer to the object we are the more important it becomes to follow the desired trajectory.

This property is reflected in the planner by adding a higher weight (γ) to the trajectory-following dynamics when we get closer to the target. And in reverse, a long distance to the target leads to a higher weight (β) to the goal directed dynamics. To follow a specified path to the goal a low value of β and a high value of γ is needed, see equation 4.16. The weights β and γ are variables

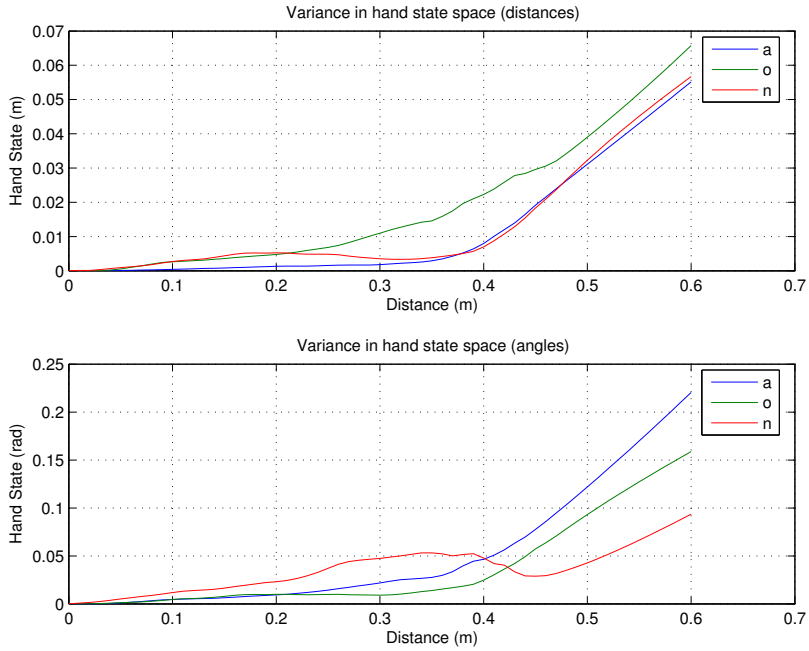


Figure 4.5: *Position- and orientation-variance of the hand-state trajectories as function of distance, across 22 demonstrations of a reaching to grasp motion. Note that distances over 0.47 are extrapolations made by the clustering method.*

reflecting the importance of the goal versus the path acquired from the variance. How they change over time in a typical reaching task is shown in figure 4.6. In the experiment in this section, β is set to $d^{robot}(t)/d^{robot}(t_0)$, and γ to $3 * (1 - \beta)$ based on the statistics from figure 4.5, where d^{robot} is obtained from equation 4.14. Recall that the position of the object must be known to compute the hand-state, so in this section we use the final position of the hand for this purpose. Therefore, the variance at zero distance is zero.

The controller has a feedforward structure as in figure 4.7. The reason for this structure is that a commercial manipulator usually has a closed architecture, where the controller is embedded into the system. For this type of manipulators, a trajectory is usually pre-loaded and then executed. Therefore, we generate the trajectories in batch mode for the ABB140 manipulator. Since our approach is general, for a given different robot platform with hetroceptive sensors (e.g., vision) our method can be implemented in a feedback mode, but this requires that the hand-state $H(t)$ can be measured during execution.

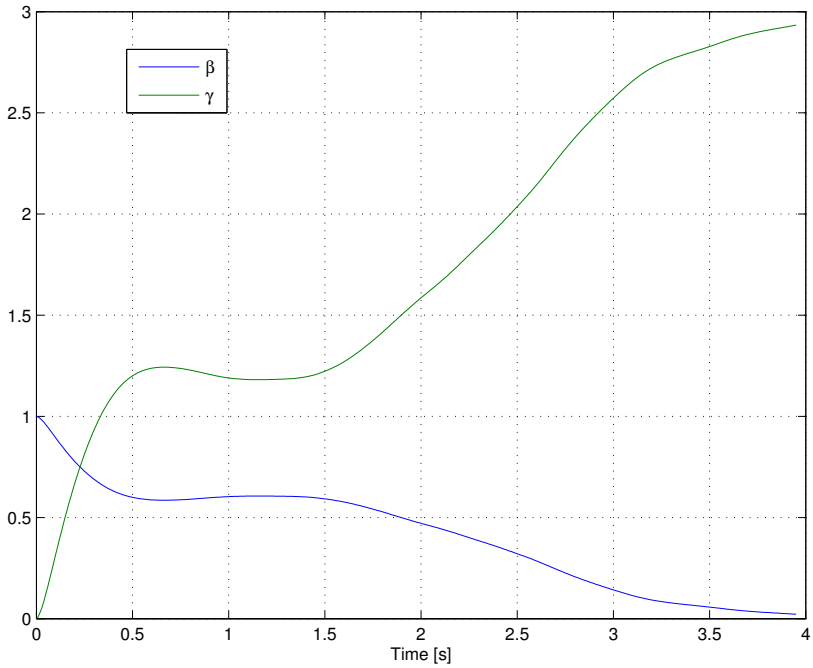


Figure 4.6: *The weights β and γ determining the trade-off between following a trajectory or moving towards a target. These are determined from the normalized distance to the target.*

4.2.3 Experimental Evaluation

To illustrate our approach a set of reaching motions is demonstrated and recorded. In this section, robotic simulations are made in MATLAB using "Robotic Toolbox" [Corke, 1996]. Two experiments are performed:

1. Generate a new trajectory based on one of the models from the recorded trajectories but in the robot's workspace.

2. Generate trajectories from the robot's home position to 100 randomly placed objects within the reachable workspace of the robot.

As experimental platform we used a motion capturing device for the human demonstration and an industrial serial manipulator (ABB IRB140) equipped with a two-finger gripper. In these experiments the manipulator is simulated.

Capturing the Human Demonstration

For motion capturing of human demonstrations we use the Immersion CyberGlove with 18 sensors to capture the hand configuration. The glove contains

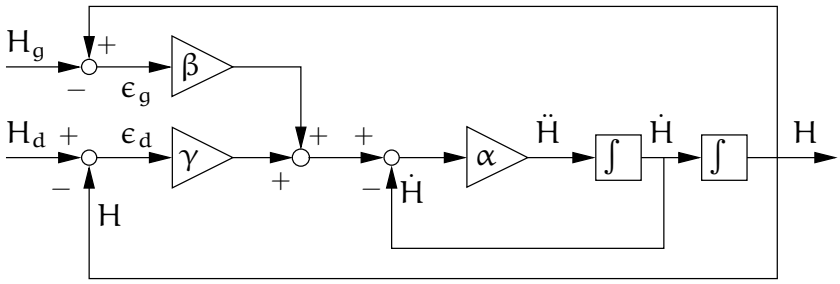


Figure 4.7: *Hand-state planner architecture.* H_g *is the desired hand-state goal,* H_d *is the desired hand-state at the current distance to target.*

strain gauges placed on the fingers, between the fingers, and on the palm and wrist. Each sensor measures a bend angle at approximately 12 Hz. In the experiments the yaw and pitch angle of the wrist are used to determine the position of the grasp center point; the thumb abduction sensor is used to determine *if* a grasp is performed. The remaining sensors measure joint angles of the fingers, which can be used for grasp recognition but are ignored in these experiments.

A Polhemus FASTRAK 6 DOF magnetic tracker is mounted on the wrist of the glove to capture the position and orientation of the hand, see section 3.3.5 for more details. T he experimental setup is shown to the left in figure 4.8. Compared to vision systems the use of wearable sensors is somewhat impeding for the user but provides several important benefits:

1. Wearable sensors are generally more accurate.
2. They do not suffer from occlusion.
3. Typically they provide fast and simple computation.
4. Ambiguities can be avoided by placing sensors directly on the desired point of interest.

In the experiments, a demonstrator performs a set of grasps of a cylindrical object (bottle) located at a fixed position. The grasp begins from an arbitrary position in the workspace. The demonstrator's arm/hand moves toward the object and the grasp ends when the object is firmly caught in the hand. Then, the hand moves away from the object and rests before repeating the task from another initial position. The motions are automatically segmented into reach and retract motions using the velocity profile. In our experiments only reaching motions is considered. A set of paths for hand motions is shown to the right in figure 4.8.

The poor performance of the magnetic tracker in the near vicinity of a real robot required the demonstration to be performed elsewhere. Therefore, we captured the data from the demonstration at a different location and then transformed it into the coordinate frame of the robot.

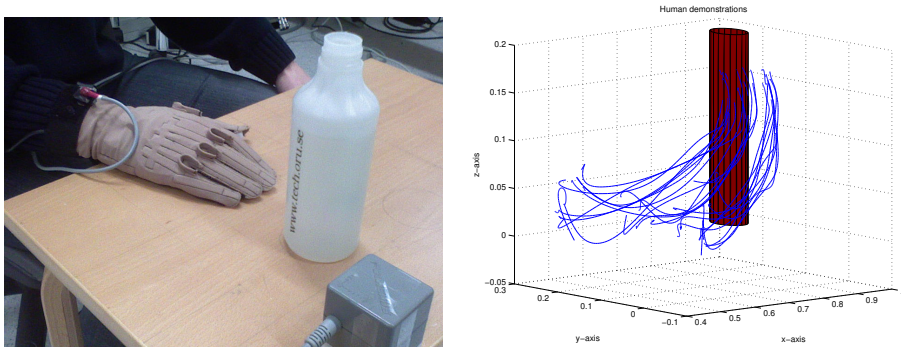


Figure 4.8: Left: *The experimental setup with the data glove and the 6D tracker used for data collection.* **Right:** *Trajectory of a reaching and grasp task. 24 demonstration reaching for a cylindrical grasp.*

Experiment 1 – Hand-State Reconstruction

In the first experiment, we try to generate robot trajectories from the robot's initial positions, which are different from the demonstrator's initial position. It is assumed that the final position is the same for both the robot and the demonstrator. A set of 24 demonstrations is used (shown in figure 4.8), where two were discarded by the automatic segmentation algorithm. Statistics from all remaining 22 demonstrations are used to make the variance calculation, shown in figure 4.5, and the coefficients a and b in Fitts' law (see section 2.2.2). Then, each of the trajectories, $H(t)$, is tested for robot trajectory generation. The final position of the path is the same for both the robot and the demonstration, but the initial positions are different. In figure 4.9 the hand-state trajectories from three demonstrations are displayed together with the corresponding generated trajectory for the robot. By comparing the hand-state trajectory of the demonstrator the one of the robot we can see how similar they are. As expected, we can see in the graphs in figure 4.9 that, despite the different initial conditions, the generated trajectory has converged to the demonstration after approximately $1.5 - 2.0$ sec. The graphs at the bottom of figure 4.9 show the hand-state component $\phi_f(t)$ describing the angle between the index finger and the thumb (a normalize value across the demonstrations, not the actual angle). Here, $\phi_f(t)$ is used as a simple grasp recognition feature, where 1 corresponds to "open" and 0 means "closed". Since we don't consider grasping in this experiment, $\phi_f(t)$ is only used to illustrate how the hand-state can be used to synchronize reaching with grasping. Figure 4.10 shows the generated and demonstrated reaching motions towards objects for four recorded trajectories.

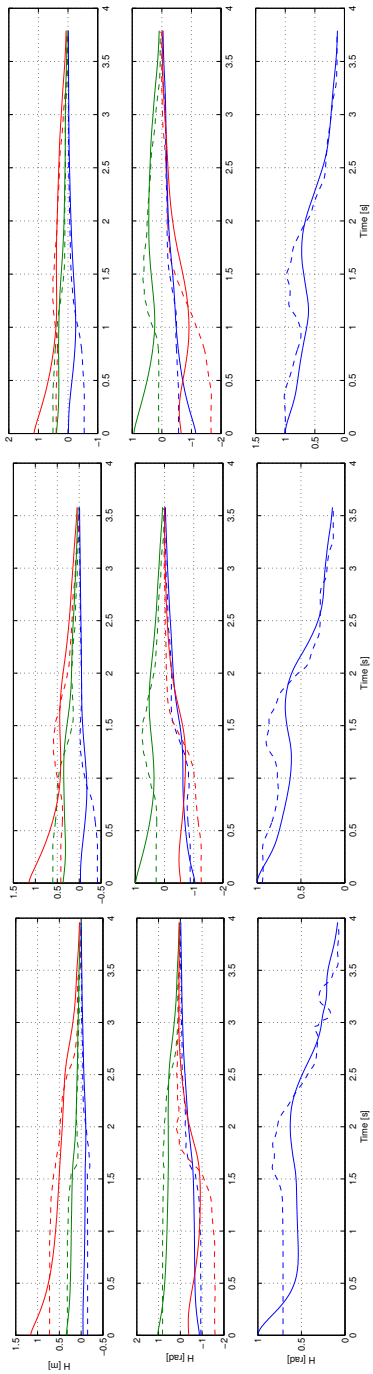


Figure 4.9: *Three sample demonstrations and the corresponding imitations. Solid lines are trajectories produced by the controller. Dashed lines are recorded from the demonstration. In the top and middled graphs, blue is distance and rotations in n-axis, red o-axis, and green a-axis. The bottom graphs are the band-states of the gripper.*

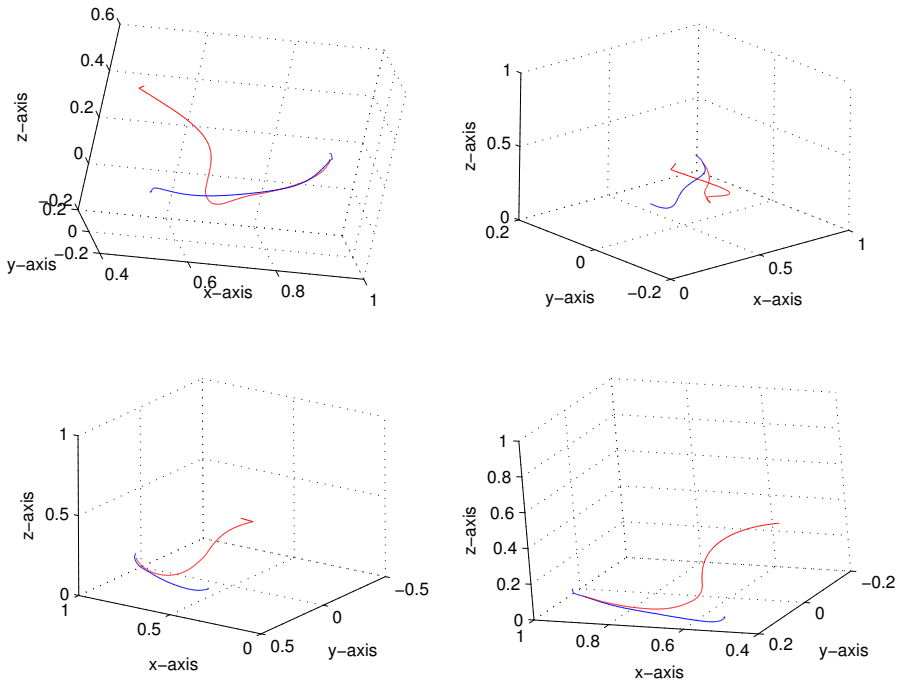


Figure 4.10: *Four sample trajectories. The blue line is trajectory executed by the robot, the red the demonstrated trajectory to follow. (The object is not shown here to not occlude the path.*

Experiment 2 – Trajectory Generalization

In the second experiment trajectories are generated toward objects placed at random positions within the reachable area of the workspace. The robot started from the same initial pose (home position where the position of the joints are: $\Theta = [0, 0, 0, 0, 0, 0, 0]$), and one trajectory–one for each of the 22 models–was generated to each of the 100 randomly generated positions of the object. This means that in total 2200 trajectories were tested. It is assumed that these objects have observable affordances acquired a priori by some other means; e.g., vision, intelligent environment or CAD models. In total, 1425 of 2200 trajectories have been classified as "successful". Success is defined as:

$$t_{reach} < 1.5 * t_{est} \qquad \text{and} \qquad \sqrt{d_f^2 - d_{des}^2} < 0.01m \tag{4.18}$$

where

No.	1	2	3	4	5	6	7	8	9	10	11
SR	100%	99%	99%	99%	99%	31%	99%	99%	8%	7%	42%
Var	0.08	0.06	0.16	0.03	0.11	0.77	0.22	0.11	0.30	0.95	0.45
No.	12	13	14	15	16	17	18	19	20	21	22
SR	99%	99%	94%	87%	5%	1%	13%	79%	15%	99%	52%
Var	0.02	0.12	0.20	0.12	1.08	0.00	0.48	0.17	0.18	0.42	0.30

Table 4.1: *The success rate for each model by which they generated a successful reaching motion. "SR" is the success rate, and "Var" is the variance of the duration.*

t_{est} is the estimated duration of the movement,
t_{reach} is the actual reaching time
d_{des} is the desired distance to the object at the end of the motion
d_f the actual end-effector distance to object at the end of the motion

In this experiment $d_{des} = 0$, since the position at t_f defines the base of the hand-state space. In table 4.1 the success rate and the variance for each model are shown. Here "SR" means success rate and "Var" is the variance of movement duration. Model 1-5, 7, 8, 12, and 13 succeeded in reaching the target 99-100% of the time and with a low variance. However, model number 9, 10, 16, and 17 succeeded in less than 10% of the trials and have a high variance. The model number 21 has a high success rate but also a high variance. Figure 4.11 show the success-rate and the variance in a graph where the models in the lower right corner are the ones with low variance and high success-rate.

The variance in hand-state over the 1425 trials is shown in figure 4.12. As a reference, the duration of the human demonstrations had a mean value of 3.22 sec and a variance of 0.33 sec.

By testing the performance of each model, it can be decided if it generalizes satisfactorily enough or not. Models that fail to generalize can be removed since all demonstrations describe the same type of task. Furthermore, the generalization capability can be used as a performance metric that can be used in a reinforcement learning framework, as we will see in chapter 5.

4.3 Next-State-Planner, Simplified Version

The next-state-planner uses the demonstration to generate a similar hand-state trajectory with the distance as a scheduling variable. Hence, the closer to the object the robot is the more important it becomes to follow the demonstrated trajectory. This property is reflected by adding a higher weight to the trajectory-following dynamics when we get closer to the target. In revese, long distance to the target leads to a lower weight to the trajectory following dynamics.

The final position of the path is the same point as the target position, thus, the point of attraction becomes the same at the end of the motion. Therefore, the planner can be simplified if the trajectory following property is judged to

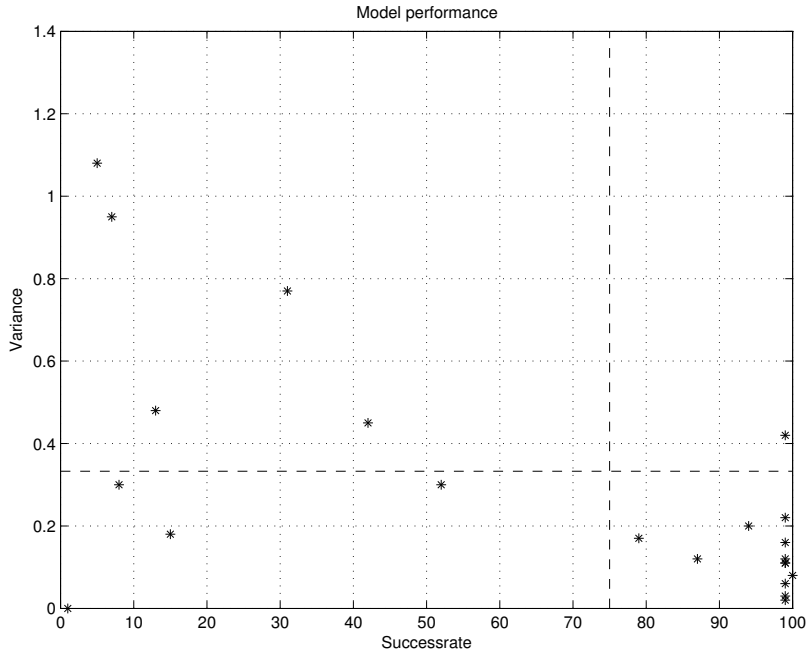


Figure 4.11: *The performance of each model measured by the success-rate and the variance. The horizontal line is the human variance in movement duration, and the vertical line is an ad-hoc separation in success-rate.*

be of greater importance, thus, the β weight is set to zero. Hence, system (4.16) gets the following dynamics:

$$\ddot{H} = \alpha(-\dot{H} + \gamma(H_d - H)) \tag{4.19}$$

where $H, \dot{H}$ and $\ddot{H}$ are the hand-state, and its first and second derivatives. H_d is the desired hand-state trajectory encoded in equation 3.16, α is a positive gain and γ is a positive weighting parameter for the importance of the tracking term. A block scheme of the planner is shown in figure 4.13.

The weight γ reflects the importance of the path following capability of the planner and is acquired form the variance in multiple demonstrations, see section 4.2.1. We have empirically found γ to produce satisfactory results for:

$$\begin{aligned} \gamma_{pos} &= 0.3\frac{1}{\sqrt{Var(H_{xyz}(d))}} \\ \gamma_{ori} &= 5\frac{1}{\sqrt{Var(H_{rpy}(d))}} \end{aligned}$$

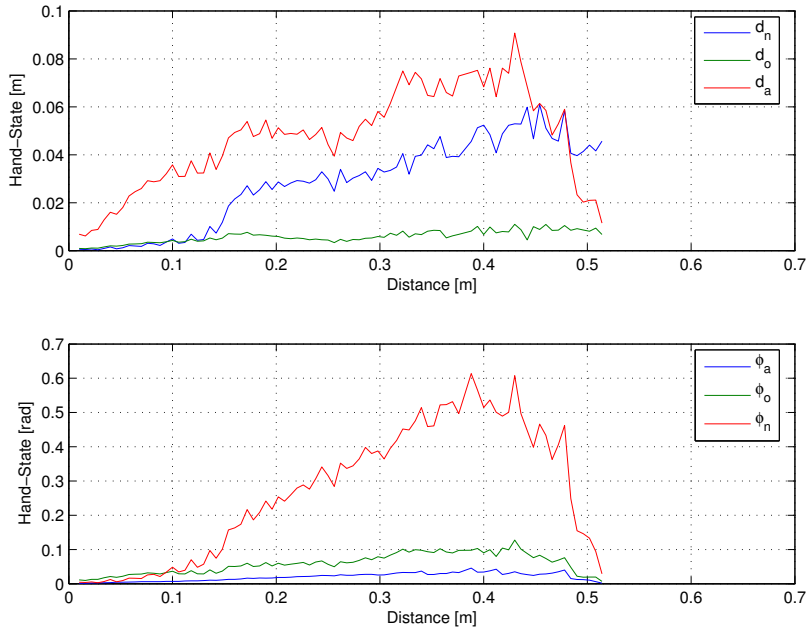


Figure 4.12: *The variance over 1425 trajectories where the models succeeded in reach-to-grasp.*

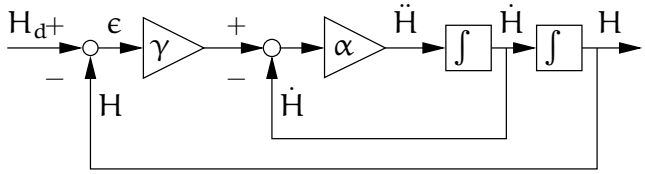


Figure 4.13: *Hand-state planner architecture.* $\mathrm{H_d}$ *is the desired hand-state at the current distance to target.*

where γ_{pos} and γ_{ori} are the weights for position and orientation, respectively. From equation 4.15 $\mathrm{Var}(\mathrm{H_{xyz}(d)})$ and $\mathrm{Var}(\mathrm{H_{rpy}(d)})$ are the variances for the position and orientation, of the respective hand state component, see figure 4.14. In our experiments α_{pos} and α_{ori} were fixed at 8 and 10, respectively, with the time difference between two points in the generation being $\mathrm{dt} = 0.01$. These gains were chosen to provide dynamic behavior similar to the demonstrated motions, but other criteria can also be used.

Analytically, the poles in equation 4.19 are found from:

$$\mathrm{p_1, p_2} = -\frac{\alpha}{2} \pm \sqrt{\frac{\alpha^2}{4} - \alpha\gamma} \tag{4.20}$$

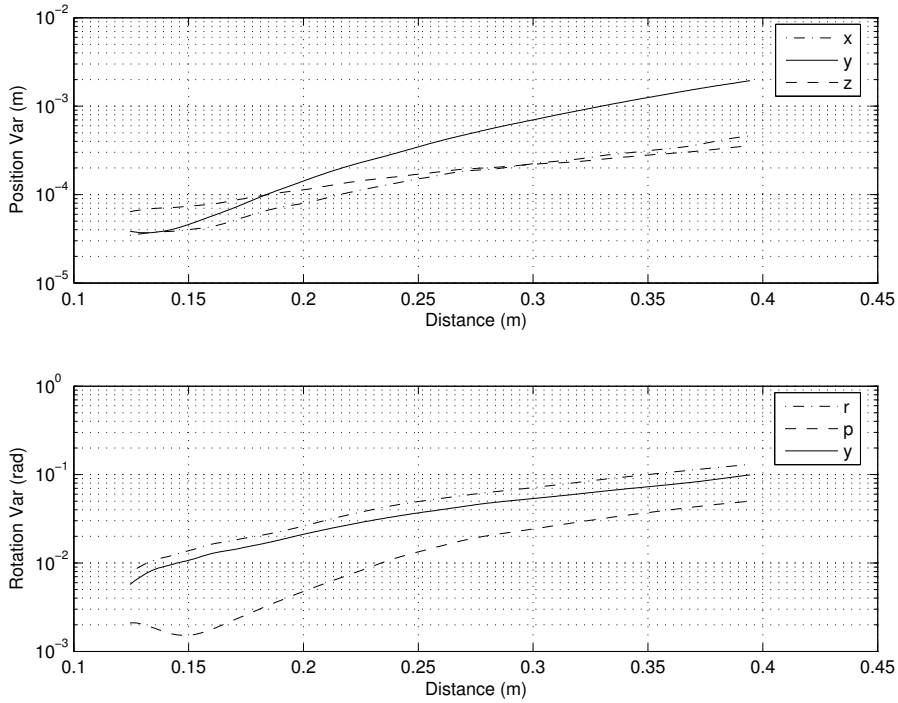


Figure 4.14: *Position- and orientation-variance of the hand-state trajectories as function of distance, across 21 demonstrations of a reaching motion to grasp a soda can.*

so the real part of p_1 and p_2 will be $\leqslant 0$, which will result in a stable system [Levine, 1996]. Moreover, $\alpha \not\leqslant 4\gamma$ and $\alpha \geqslant 0$, $\gamma \geqslant 0$ will contribute to a critically damped system, which is fast and has small overshoot. Figure 4.15 shows how different values γ affects the dynamics of the planner in a tracking task.

In these experiments the hand-state does not contain any hand configuration components. The hand-state is defined to contain six hand-object relation components: displacement x, y and z direction and rotation around the three axes: roll r, pitch p and yaw y, see figure 4.1.

4.3.1 Experiments

For these experiments human demonstrations of a pick-and-place task are recorded with two different subjects using the PhaseSpace Impulse motion capturing system described below. The motions are automatically segmented into reach and retract motions using the velocity profile and distance to the object. The robot used in the experiments is the industrial manipulator ABB IRB140. The robot is equipped with an anthropomorphic hand developed at the Royal

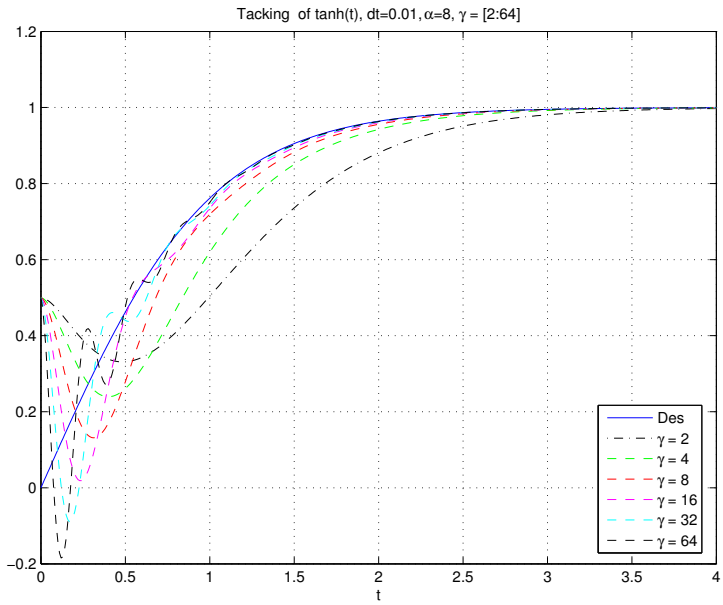


Figure 4.15: *The dynamics of the planner for six different values of* γ*. The tracking point is* $\tanh(t)$*, with* $dt = 0.01$ *and* α *is fixed at* 8.

Institute of Technology (Stockholm, Sweden), described in detail by Tegin et al. [2008].

Motion Capturing System

The Impulse motion capturing system consists of four cameras mounted around the operator to register the position of the LEDs. Each LED has a unique ID by which it is identified. Each camera can process data at 480 Hz and have 12 Mega pixel resolution resulting in sub-millimeter precision. The Impulse systems can be seen in figure 4.16. The operator wears a glove with LEDs attached to it, see figure 4.16 and 4.17. Each LED modulates at a unique frequency giving them a unique ID. Thus, each point on the glove can be associated with a finger, the back of the hand or the wrist. The LEDs on top of the palm are used to compute the orientation of the hand. One LED is mounted on each finger tip, and the thumb has one additional LED in the proximal joint, see figure 4.17. One LED is also mounted on the target object.

Experiment 1 – Gripper Pose Variation

To investige how end-effector position–and hence the approach trajectory–affect grasp success, we have dynamically simulated the grasping of an or-

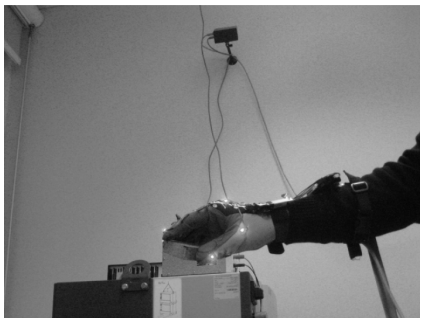


Figure 4.16: *The Impulse motion capturing system from PhaseSpace.*

Figure 4.17: *The Impulse glove.*

ange using different hand positions across a 3D-grid. All experiments were performed using the GraspIt! simulator. The hybrid force/position controller is applied to the Barrett hand model from grasp initiation until the grasp is completed. See Tegin et al. [2008] for a detailed description of the robotic hand and the hybrid force/position controller. A successful grasp can be either a precision disc grasp or a power grasp. A grasp is considered to have failed if no force closure grasp were reached during grasp formation. Figure 4.18 shows the results from such simulations. Additional simulations and control details can be found in e.g., Tegin et al. [2009].

These experiments showed that the required position accuracy of the robot hand is in the centimeter range. This means that the reaching motion must position the the end-effector with an accuracy within the range required by the anthropomorphic hand. The variance in the robot trajectories, shown to the left in figure 4.19, is within the millimeter range which satisfies the accuracy requirements. The required accuracy of the reaching motion is depended of capabilities of the gripper; an autonomous robotic hand like the Barrett hand or the KTHand impose a looser constraint on the reach motion than a parallel gripper, which requires much higher accuracy. For fully autonomous execution of a grasp learnt using the suggested approach, we must also consider uncertainties with respect to object position, orientation, and in the object model itself.

Experiment 2 – Learning from Demonstration

For this experiment 26 demonstrations of a pick-and-place task were performed. A soda can was grasped with a spherical grasp. To make the scenario more realistic the object is placed with respect to what is convenient for the human and what *seems* to be feasible for the robot.

Five of the 26 demonstrations were discarded in the segmentation and modeling process for reasons such as failure to segment the demonstrations into

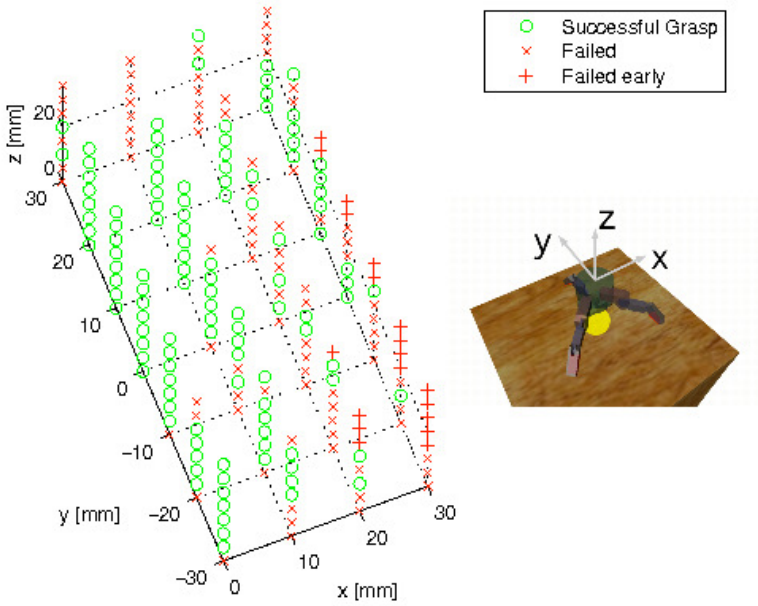


Figure 4.18: *Grasp results from dynamic simulations of different initial hand positions. Grid spacing is 10 mm in the xy-plane and 5 mm along the z-axis. Courtesy of Johan Tegin.*

three distinct motions (approach, transport and retract) or the amount of data was not enough for modeling because of occlusions. In this experiment, only the reach-to-grasp phase of the motion is considered. All 21 demonstrations were used for trajectory generation and to compute the variance, shown in figure 4.14. The trajectory generator produced 21 reaching motions which are loaded to the robot controller and executed. Note that, for each produced trajectory, all demonstrations are used to compute the γ-gain, which determines how much the robot can deviate from the followed trajectory. In eight attempts, the execution succeeded while 13 attempts failed because of unreachable configurations in joint space. This could be prevented by placing the robot at a different location with better reachability. Moreover, providing the robot with more demonstrations, with higher variations in the path, will lead to fewer constraints. Three samples of hand-state trajectories of the successfully generated ones are shown in figure 4.20.

In the eight successfully executed reaching motions we measured the variation in position of the gripper, shown to the left in figure 4.19, which is within the millimeter range. This means that the positioning is accurate enough to en-

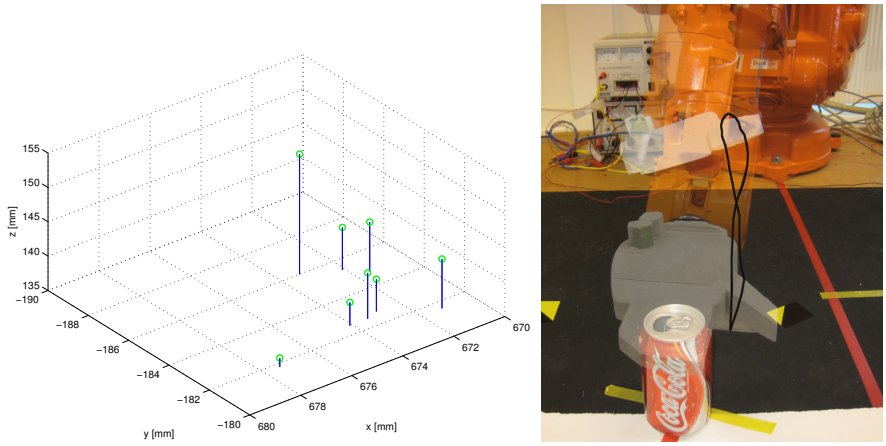


Figure 4.19: Left: *The end-effector position in the workspace for the eight successfully executed trajectories.* **Right:** *A trajectory generated when the initial position is the same as the desired final position, showing that the method generate trajectories as similar to the demonstration as possible based on the distance.*

able successful grasping using an autonomous gripper, such as the Barrett hand [Tegin et al., 2007] or the KTHand.

Experiment 3 – Generalization in Workspace

In this experiment, the method is tested on how well it generalizes by examining if feasible trajectories will be generated when the object is placed at arbitrary locations and when the initial configuration of the manipulator is very different from the demonstration. This will determine how the trajectory planner handles the correspondence problem in terms of morphological differences.

If the initial distance between the end-effector and the target is outside the data range, the TS-models must be extrapolated, a risky strategy for longer distances. Another approach is to apply a different control scheme for this region, e.g., the VITE strategy [Bullock and Grossberg, 1989] if the distance is within the data range the proposed trajectory generator takes over.

Three tests were performed to evaluate the trajectory generator in different parts of the workspace.

- First, trajectories are generated when the manipulator's end-effector starts directly above the object at the desired final position with the desired orientation that is $H_1^r = H_f^r$. The resulting trajectory is shown to the right in figure 4.19. Four additional cases are also tested displacing the end-effector by 50 mm in +x, -y, +y, and +z direction from H_f, all with very similar results (from the robot's view: x is forward, y left and z up).

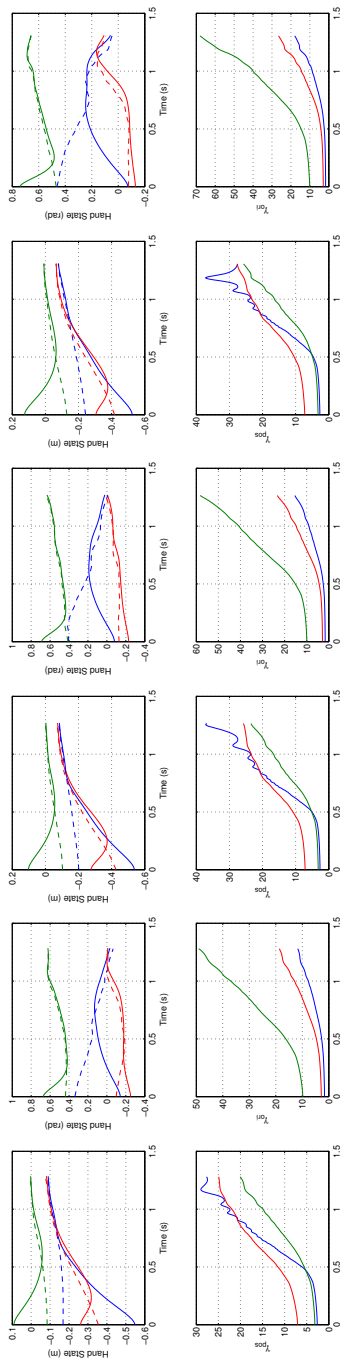


Figure 4.20: *Three samples of demonstrations, and the corresponding imitations. The top graphs shows the hand state trajectories, position and orientation for each of the demonstrations/imitation. The solid lines are the trajectories produced by the planner, dashed lines are the model constructed from the recorded demonstration.*

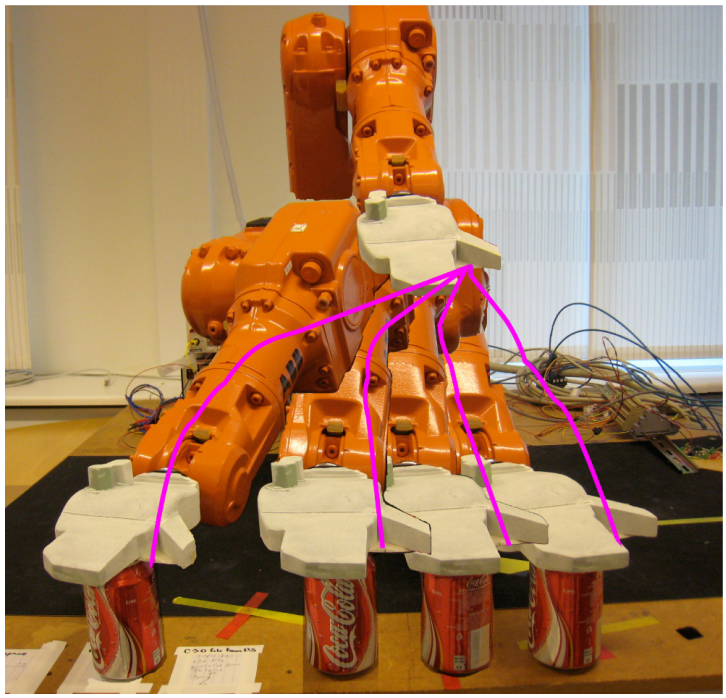


Figure 4.21: *The object is placed at four new locations within the workspace.*

- Second, we tested reaching the object at a fixed position from a random initial configuration. Figure 4.22 shows the result from two random initial positions where one trajectory is successfully followed and the other one fails. The failure is a result of operation in hand-state space instead of in joint space, and it might therefore have a tendency to go onto unreachable joint space configurations, as seen in the right column of figure 4.22. To prevent this it is possible to combine two controllers: one operating in joint space and the other in hand-state space, similar to the approach suggested by Hersch and Billard [2006], but at the price of violating the demonstration constraints.

- Third, the object is placed at four different locations within the robot's workspace; displaced 100 mm along the x-axis, and -100 mm, +100 mm, +200 mm, and +300 mm along the y-axis, see figure 4.21. The initial pose of the manipulator is the same in all reaching tasks. The planner successfully produces four executable trajectories to the respective object position.

The conclusion from this experiment is that the method generalizes well in the tested scenarios, thus adequately addressing the correspondence problem.

However, the unreachability problem has to be addressed in future research to investigate how the robot should balance the two contradiction goals: reaching an object in its own way, with the risk of collision, and reaching an object as the demonstrator showed. Indeed, if the robot has more freedom to choose the path it is more likely to avoid unreachable configurations. However, such freedom increases the risk for collision.

Experiment 4 – A Complete Pick-and-Place Task

To test the approach on an integrated system the KTHand is mounted on the ABB manipulator and a pick-and-place task is executed, guided by a demonstration showing pick-and-place task of a box ($110 \times 56 \times 72$ mm). The reaching motion and the grasp are executed as described in the previous experiments in this section. The synchronization between reach and grasp can be performed by a simple finite state machine. After the grasp is executed, the motion to the placing point is performed by following the demonstrated trajectory (see section 3.3.4). Since the robot grasp pose corresponds approximately to the human grasp pose it is possible for the planner to reproduce the human trajectory almost exactly. This does not mean that the robot actually can execute the trajectory, due to workspace constrains. The retraction phase follows the same strategy as the reaching motion, but in reverse. Figure 4.23 shows the complete task learned from demonstration.

4.4 Summary

In this chapter, we presented a method for Programming-by-Demonstration of reaching and grasping tasks. A hand-state representation is employed as a common language between the human and the robot which allows the robot to interpret the human motions as its own. It is shown that the suggested method can generate executable robot trajectories based on current and past human demonstrations despite morphological differences, thus meeting the first goal outlined in the introduction. The second goal, the generalization ability of the trajectory planner is illustrated by several experiments where an industrial robot arm executes various reaching motions and positions the gripper accurately enough to perform a power grasp using a three-fingered hand. The third goal (synchronization between reaching and grasping), which for a simple–yet very common–type of gripper, have been shown in how the different components of the hand-state are synchronized. Since the hand-state provides a description of both the human hand and the gripper, it also gives the robot the ability to interpret its own motions in the same way as the demonstrator's. Therefore, the robot can learn from its own experience in a similar manner, the fourth outlined goal. We will elaborate this in the next chapter.

One main question for future research is how to learn from the demonstration the gains α, β and γ used to control the dynamics of the next-state-planner

(see section 4.2.2 and section 4.3). Currently they are manually determined, even if the variance is incorporated in their choice.

Reinforcement learning, discussed in chapter 5 can be used to make the robot improve over time if it has access to a performance metric. This is a key question in future research.

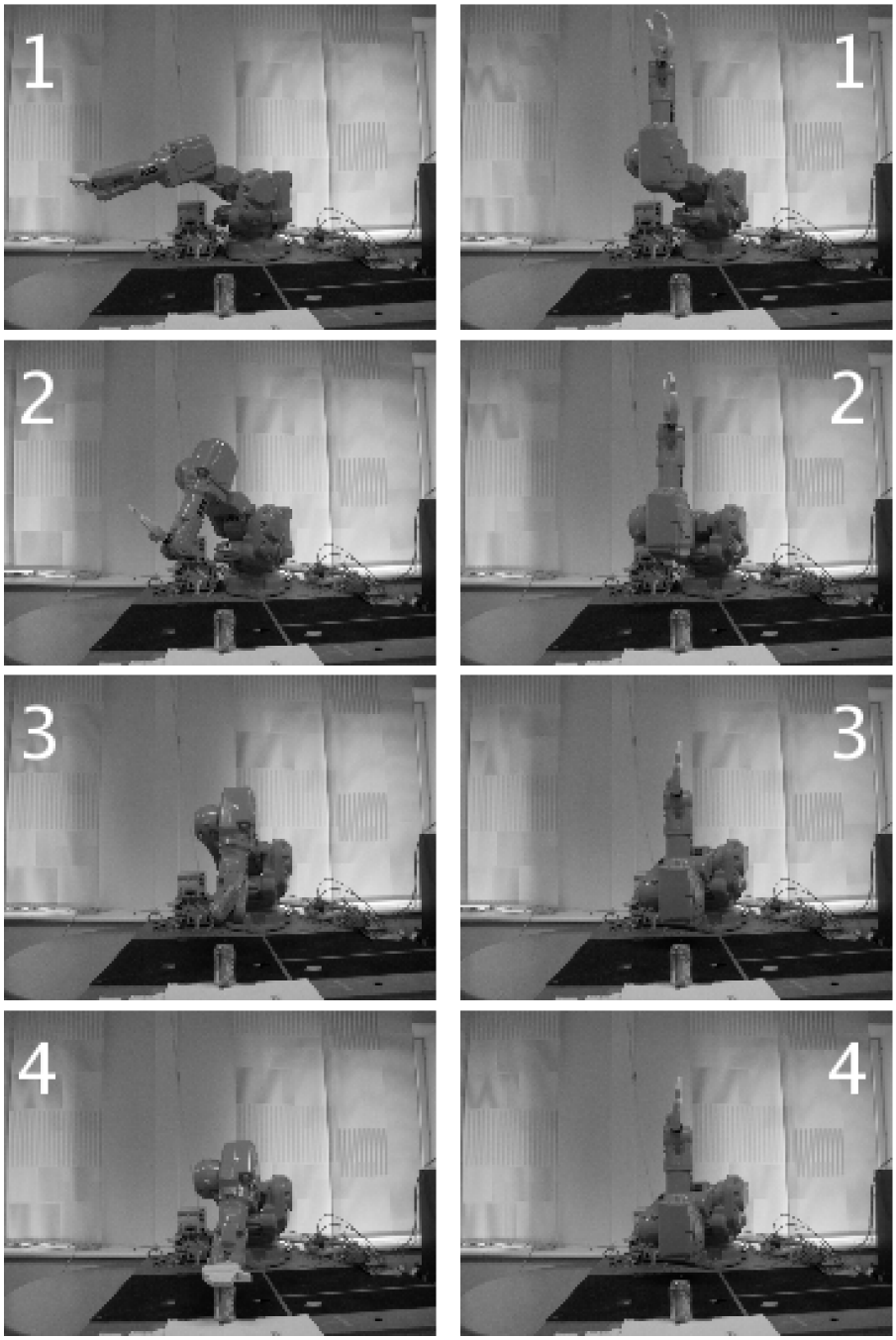


Figure 4.22: *A trajectory generated from two different randomly initial position reaching for the same object. In the left column, a successful reaching motion is generated where the final position is on top of the can. The right column shows a case where the robot reaches an unreachable joint configuration and cannot move along the trajectory.*

Figure 4.23: *Industrial manipulator programmed using a demonstration. A movie of the sequence is available at: http://www.aass.oru.se/Research/Learning/arsd.html.*

Chapter 5
Reinforcement Learning for Reaching Motions

In this chapter we investigate how reinforcement learning can be used in a Programming-by-Demonstration framework, and how reinforcement learning can benefit from a demonstration in order to reduce the search space. When a robot has learned how to perform a task, it should be able to self-improve its skills and learn new skills.

The contributions of the experiments in this chapter are as follows:

1. We show how demonstrations speed up reinforcement learning of reaching motions.
2. We present a reinforcement learning strategy for the fuzzy time-model based next-state-planner. This learning strategy enables the robot to evaluate its performance and adapt its actions to better fit its own morphology instead of following a demonstration.

Section 5.1 briefly introduces reinforcement learning, and section 5.2 and 5.3 present the experiments on reinforcement learning in a Programming-by-Demonstration setting. A short summary and a discussion are given in section 5.4.

5.1 Reinforcement Learning

Reinforcement learning is very general in its formulation and can cover a wide variety of problems [Sutton and Barto, 1998]. As a practical example, consider an agent (such as a robot) that learns a control policy during its interaction with the environment. From a given state s, the agent takes an action a, following policy π, or to be formal:

$$\pi(s) \rightarrow a \tag{5.1}$$

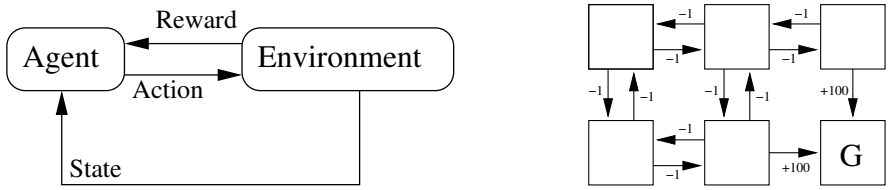


Figure 5.1: Left: *The interaction between the agent and the environment.* **Right:** *At each transition a reward of* -1 *is given for all actions, except the ones leading to the goal state denoted G. The transitions to the goal state give a reward of* $+100$*, and the trial is terminated.*

Initially it is assumed that the agent has no knowledge about the world it operates in, but gains knowledge from exploring the world with initially random actions. During the exploration of the environment the agent receives a response from the environment in the form of a state transition and a reward for each transition, given a certain action, illustrated in figure 5.1. The agent's objective is to maximize the accumulated reward, meaning that the reward function must indicate the goal of the system. The reward accumulation is done by summing the rewards over time:

$$R = r(s_0, a_0) + \ldots + r(s_n, a_n) \tag{5.2}$$

where $r()$ is the reward function for a state-action pair s_n, a_n at time step n and R is the total reward.

Consider the example to the right of figure 5.1. A positive reward is assigned when the agent reaches the goal, and a small negative reward is given for every (time) step the agent spends searching for the goal, hence the reward of -1 for all other state transitions. The positive reward the agent receives when performing an action is often delayed until a goal is reached. A delayed reward usually means that the agent receives the same reward everywhere (for example 0 or -1) until the goal state or a forbidden state is reached, where a large reward or punishment is received. This way of assigning rewards will result in a system that strives towards reaching the goal state as quickly as possible. If the reward is given for the goal state and not for other states that are considered by the designer to be good, reinforcement learning solutions tend to converge towards the optimal solution, although this is not always guaranteed. When using delayed rewards the reinforcement learning agent has in its first trial(s), by luck or accident, to run into the goal state before it can begin to perform better than the initial random behavior.

If reinforcement learning is compared to supervised learning, where a target function explicitly shows how to perform a task, the reinforcement learning agent is only provided with a quality measure like "good" or "bad", but not

with the input-output mapping as in supervised learning. Hence, reinforcement learning is designed to express *what* to do, without having a teacher telling you *how* to do it.

If the agent keeps the exploration capability it is possible for the agent to learn a better policy than the one initially learned, assuming that it was not optimal.

Generally, a task can be considered to belong to one of two cases, either it is an episodic task or a continuing task. An episodic task is an activity that *ends*, like a reaching motion, that *trial* is over and the next one starts. The other case is a continuing task, when the agent is not trying to get to the goal but rather "keep close to it".

5.1.1 Temporal-Difference Learning

One of the major developments within reinforcement learning was Temporal-Difference learning (TD-learning) by Sutton [1988]. TD-learning waits only until the next time step to update the value function. Given state-action values, denoted by $Q(s_t, a_t)$, the temporal difference between time t and $t+1$ is of interest:

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{5.3}$$

where δ_t is the TD-error at time t and γ is the discount factor. The update equation using the TD-error is then:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\delta_t \tag{5.4}$$

where α is a step size parameter. One of the most popular TD-learning techniques is Q-learning which therefore deserves some more attention.

5.1.2 Q-learning

Q-learning was originally developed by Watkins [1989] and has become one of the most popular learning algorithms within the reinforcement learning family. In Q-learning each state-action pair is marked with a quality measure. For value estimation and control, two different functions are used, Q^π and π, respectively. When different functions are used for estimation and control the method is called an "off-policy method".

The action selection policy can either be greedy, which means that the action with the highest value is selected, or ϵ-greedy that means with small probability ϵ a random (i.e., exploring) action is selected. With a low value of ϵ the exploiting action is more often selected. When a policy π is followed, the Q-function is given a value by:

$$\pi \leftarrow Q^{\pi}(s, a) \tag{5.5}$$

The ongoing process of self-evaluation that the agent continuously goes through contains both evaluation and improvement of the policy. When an action is selected and taken, an evaluation is given in response to this action by a reward signal. This reward is then used to update the Q-value.

In the discrete form of Q-learning, the Q-value $Q(s, a)$ is usually implemented as a table. To update the Q-value for each state-action the following equation is used iteratively during learning:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{5.6}$$

where s is the state, s' next state, a action, a' next action, α step size parameter, r the reward and γ a discount factor. Note that the Q-learning algorithm is recursive, and a starting value is needed. Usually this is initialized to zero, although it could be initialized arbitrarily. The full Q-learning algorithm is shown in table 5.1.

If the agent is shown how to perform a task by a teacher, it can record the actions and update the Q-table during the demonstration. The RL-agent will then start with a Q-table containing non-zero values in the state-action space where the teacher provided examples, which will make the agent to follow the teacher's example, unless an exploring action is selected. Hence, the policy is biased by the demonstration, and initially performs better (or at least differently) from a random behavior. If the ϵ-parameter, which determines the randomness of the actions, is set to zero the agent would always exploit the Q-table by selecting the maximum reward action, hence perform exactly as the teacher does in a deterministic environment. However, when setting the ϵ-parameter to some small number the agent can explore and eventually find a better solution than the teacher, if it exists.

5.1.3 Large Continuous State- and Action-Spaces

Reinforcement learning was developed for discrete problems, hence the type of problems reinforcement learning can be applied to was limited to discrete formulations. Therefore, several researchers have worked to extend reinforcement learning to continuous values, e.g., [Doya, 2000, ten Hagen, 2001].

In the case of large state- and action-spaces, a generalization must be done, since it will be impossible to try all possible cases. If a continuous state or action problem is encountered, this means finding the $V(s)$ or $Q(s, a)$ value for the continuous case, a function approximator can be used to approximate the V- or Q-function. Neural networks are one possible method for approximating the value function, where only the weights of the network need to be stored.

- Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in S$ and $a \in A(s)$
- Do forever:
  - $s \leftarrow$ current state
  - $a \leftarrow \epsilon$-greedy(s, a)
  - Execute action a; observe the resultant state s' and reward r
  - $Q(s, a) \leftarrow Q(s, a) + \alpha\,[r + \gamma\ max_{a'} Q(s', a') - Q(s, a)]$
  - $s \leftarrow s'$

Table 5.1: *Pseudo code for the Q-learning algorithm.*

A function approximator permits representation of value functions with less variables, instead of keeping a table of all possible states and actions. A function approximator also enables generalization (interpolation and possibly some extrapolation) on previous experiences.

Reinforcement learning has been shown to perform well on mobile robot tasks with low dimensional state- and action-spaces, but when the dimensionality grows it scales poorly. Several papers address the issue of reinforcement learning in mobile robotics, normally using 2 DOF, which means a much smaller action space than that of a manipulator or a humanoid robot. Locally weighted learning is well suited for performing function approximation in high dimensional spaces, and has also proved to be useful for online incremental learning [Vijayakumar and Schaal, 2000]. By using a feasible function approximator the value function can be approximated in regions not yet experienced, which also can speed up the learning process.

When moving into higher dimensional spaces the ϵ-greedy policy becomes dangerous and the risk of failure increases. This is because the max operator, usually used to determine the next action, can make the policy evaluation unstable [Peters et al., 2003]. Increases in dimensionality require a more feasible method than the max operator as a policy gradient method to determine the best policy. This have led to the development of gradient-based methods suitable for continuous valued functions such as the vanilla gradient improvement [Gullapalli, 1993] and the natural policy gradient improvement [Peters et al., 2003].

5.1.4 The Dyna Architecture

An important contribution to reinforcement learning and artificial intelligence is the Dyna architecture, developed by Sutton [1991], which combines learning and action with planning. Next follows a description of how the Dyna architec-

- Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in S$ and $a \in A(s)$
- Do forever:
  - $s \leftarrow$ current state
  - $a \leftarrow \epsilon$-greedy(s, a)
  - Execute action a; observe the resultant state s' and reward r
  - $Q(s, a) \leftarrow (s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
  - $Model(s, a) \leftarrow s', r$
  - Repeat N times:
    - $s \leftarrow$ random previously observed state
    - $a \leftarrow$ random action previously taken in s
    - $s', r \leftarrow Model(s, a)$
    - $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

Table 5.2: *Pseudo code for the Dyna-Q algorithm.*

ture can be applied to Q-learning, although Dyna can be combined with other reinforcement learning algorithms.

In the original Q-learning algorithm the update of the Q-value is made one time per iteration, making convergence of the Q-values slow. One way to overcome this drawback is to use the so called "eligibility traces", which is a trace that the agent leaves behind, to "remember" which path was taken. Another way is to keep a memory map containing samples of states (or state-action) values visited before. Dyna uses a memory, called "model", that contains information about which states have been visited, what action was taken there, what state that action resulted in and the expected reward. After each real action the agent has taken the Q-table is updated using the reward signal. Thereafter an iterative simulation phase starts with n iterations using the model, where a random state-action pair is selected from the model, and its Q-value is updated based on the recorded data. This simulation, or planning, could be called a "mental rehearsal" process, thus implementing a so-called indirect reinforcement learning algorithm. The Dyna-Q algorithm is described in table 5.2.

5.1.5 Robotic Applications Using Reinforcement Learning

To investigate how demonstration of a task influenced the speed of the learning process, Schaal [1997] used reinforcement learning (V-learning, Q-learning and model-based reinforcement learning). An implementation on a real manipulator showed some convincing results on learning from demonstration when performing a pole-balancing task. With a demonstration the robot succeeded

in the very first trial using the model-based reinforcement learning method, which also offered more robustness in this task than the other methods investigated. He showed that model-based learning, such as Dyna, benefits greatly from demonstration and speeds up the learning process, in contrast to the other methods investigated, such as V-learning and Q-learning, which did not benefit significantly from demonstration.

5.2 A Dyna-Q Application for a Robotic Arm

This experiment addresses how demonstrations can speed up Q-learning of a reaching task. We use a concept of combining acting, learning and planning, known as Dyna (see section 5.1.4), and implement it on a simulated articulated robotic arm. We use a model of the human arm that the data of the capturing device is mapped to. The human arm model is introduced due to the inaccuracy of the learned model described in the experiment in chapter 3. In this work the aim is to use a demonstration that provides the reinforcement learning agent with knowledge of the task.

One solution to the learning problem would be to apply a direct supervised learning algorithm, where the teacher tries to provide input-output mapping examples of all possible situations by error minimization. However, this approach is problematic because the demonstrations may be incomplete and only partly correct [Kaiser et al., 1995]. Instead our approach is to apply reinforcement learning, where the demonstration may be given in order to guide and accelerate the learning through autonomous exploration.

5.2.1 Method

By using the Dyna-Q algorithm, which combines learning and action with planning, a learning agent is used to control the robotic manipulator. When the agent has made its first action and the environment has responded the Q-table and the model are updated. The model contains information about which states that have been visited, what actions were taken there, what state that action brought it to and the expected reward. When the agent has taken a real action and observed the next state and reward, it starts planning for n iterations using the model. This way an evaluation can be made of the received information about the surrounding states.

In our implementation a discrete state-space is used and the learning aims at making the manipulator reach a goal configuration from any position. There are 1331 states (each joint position is divided into 11 discrete states and three joint are used, in total 1331 states) and at each joint an action can be “go forward”, “go backward” or “stay”. In all our experiments the learning rate, α, was set to 0.1 and the discount factor, γ, to 0.95.

For reaching the goal configuration a reward of 100 was given and -1 for all other state transitions. The task is episodic, that means when the goal configuration is reached the trial terminates.

In our implementation the Dyna-Q algorithm described in table 5.2 is used. At the start the Q-table and the model are initialized with zero values. At each new trial the manipulator is initialized to a random position. If it is in the goal state the trial is terminated, otherwise an action is selected using ϵ-greedy selection from Q and s. Then the observed new state, s' and the reward are determined, and the Q-table updated accordingly. The new state s' and reward are also stored in the model. The planning then starts, where the agent selects one of the visited state-action pairs and retrieves the expected reward, and uses this to update the Q-table.

If a demonstration by the teacher is used (the first n trials, where n is the number of demonstrated trajectories) one of the recorded trajectories is replayed to the agent to bootstrap the Q-table. When all demonstrated trajectories are done, randomly initialized starting positions are given to the agent.

5.2.2 Experiment Setup

The motion capture system ShapeTape from Measurand was used in this experiment, the same as in the previously described experiment in section 3.2.1. The training is done by a human demonstrator by means of this wearable sensor device. The sensor signals of the input device are used to drive the robot's joints so that the robot copies the motion of the human arm in the most exact way possible.

5.2.3 Human Arm Model

The first step in our system was a simple imitation of the motion of the hand of the demonstrator which has already been realized by a direct connection of the ShapeTape sensor and a kinematical model of the robot. Yet it turned out that a plain imitation ignores the configuration of the demonstrator's arm during motion completely and offers no additional DOF to influence the robot's configuration. Therefore a model "in between" has been adopted that imitates the kinematics of the human arm as much as possible. The features of the human arm model are:

- Direct analytical models with kinematical parameters, for example, link lengths, ranges of angles, singularities, work spaces, numbers of DOF.
- Differential inverse kinematics to calculate the joint angles corresponding to given Cartesian coordinates [Liegeois, 1977] and [Palm, 1992].

The human arm is modeled by a 4-link mechanism with 7 DOF and the robot as a 6 DOF mechanism, see figure 5.3.

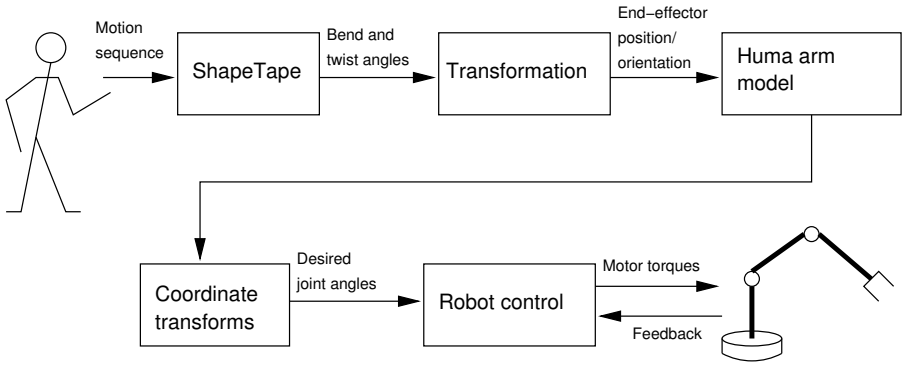


Figure 5.2: *Block scheme of the system including demonstrator, shape tape, human arm model, and robot.*

The redundancy of the human arm model and the different numbers of degrees of freedom between the human arm model and the robot requires the calculation of their direct and inverse differential kinematics, taking further into account the position and the orientation both of the human hand and the end-effectors of the robot. Figure 5.2 shows the block scheme of the whole setup.

To the left side in figure 5.4 shows a simulation of a trajectory being taught by the demonstrator using the ShapeTape. Ergonomic reasons led to the idea not simply to imitate the trajectory of the human arm but to mirror it. The method presented also makes the consideration of restrictions or side conditions possible, with respect to kinematical configurations and obstacles.

5.2.4 Simulation Results

To demonstrate the concept, a simulated robot was given the task to reach a goal configuration, shown to the right in figure 5.4, from a randomly initialized position. The simulations are done using MATLAB and a robotic toolbox [Corke, 1996], which also provides a visualization of the simulation. To make the state space convenient for calculations but without being unrealistically small the number of used DOFs is reduced from six (in conventional industrial manipulators such as the Puma 560), to only use the first three links. This permits the manipulator to reach a certain position but not a specific orientation.

The actions are deterministic, which means the manipulator will move into the next state from all states except from those that bring the joint out of range. The range span from -180 to 180 degrees for each joint.

In our experiments, it turned out that the ϵ-parameter had little influence on the learning curve. Instead the number of planning steps are of greater in-

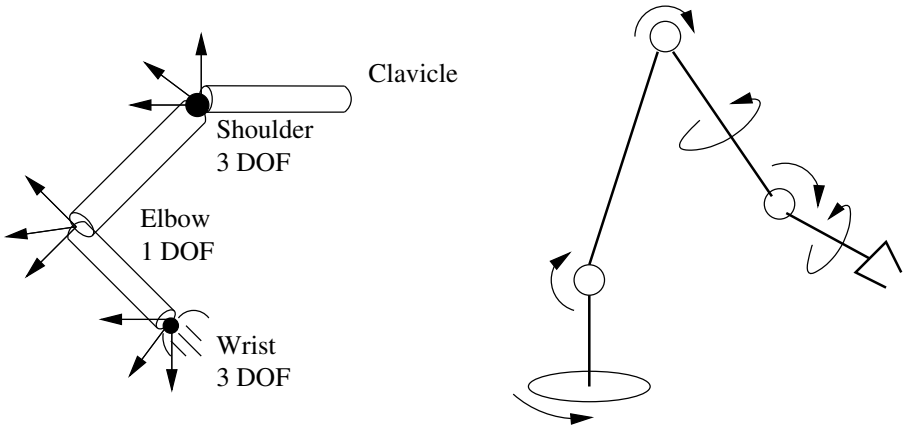


Figure 5.3: *A kinematical model of the human arm and the robot.*

terest. Comparing 0 and 100 planning steps together with a "soft planner" affects both the learning rate and the computational time. When we use 100 planning steps we call it "full planning" abbreviated FPL, and 0 planning steps we call "non-planning" abbreviated NPL. The so-called "soft planner" (SPL) is a planner that decreased the number of planning steps linearly from 100 to 0 per episode with one planning step less per episode. Six different experiments were carried out investigating the planning variable, together with the RL bootstrapped by a demonstration (denoted GU for GUided), versus RL with random exploration (denoted RA). GU and RA are combined with different planners FPL, NPL and SPL.

The left graph in figure 5.5 shows how the learning curve jumps up when the agent is "released" and explores autonomously. In this case the agent uses the demonstrated trajectories to provide the exploration policy for the first 20 episodes in order to bootstrap the learning. The non-planning agent jumps up to about 200 trials and slowly converges, while the two planning agents perform equally well, initially about 40 steps per episode. The middle graph in figure 5.5 shows initially random agents and compares the planning and non-planning cases. Here the non-planning agent, as before, needs several more episodes before converging compared to the planning agent, while the soft- and full-planner shows similar results.

A comparison between the guided and initially random case is shown to the right graph in figure 5.5 for the soft planner. Between the 20^{th} and the 35^{th} trial the initially random agent actually performs a little better than the guided agent.

Planning improves the convergence result and speeds up learning, measured in steps per episodes, but slows down learning when measuring CPU time. Even

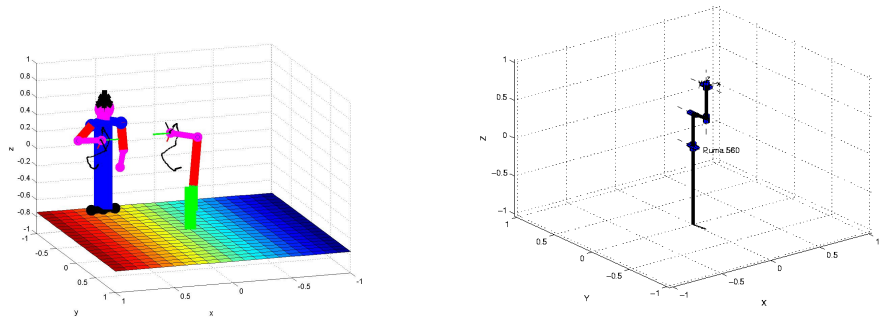


Figure 5.4: Left: *Simulation of the mapping between the wearable input device, human arm model and robot.* **Right:** *The simulated Puma560 robot in the Robotic toolbox.*

Planning mode	Guided	Initially random
Non	14.6 sec	25.4 sec
Full	708.0 sec	1435.4 sec
Soft	237.1 sec	752.0 sec

Table 5.3: *The number of seconds for each computation displayed in the figure 5.5. One hundred planning steps increases the computing time, but decreased the number of iterations before the learning converges, while no planning-step increases the number of episodes before convergence, but decreases the computational effort. The soft planner is a tradeoff of both.*

though the first trial in the unplanned tasks took approximately 2500 steps, the computational time was the lowest, about 50 times faster than its corresponding planning agent. The soft planning agent performs roughly twice as fast compared to the full planning agent. Table 5.3 summarize the comparison between the computational times needed for 0 and 100 planning steps together with the soft planner.

In these simulations the trajectories from the ShapeTape-sensor for the guided agent was not used, instead we used a set of hand-generated trajectories (20 in total). Our aim is to integrate the two modules in our future work.

5.2.5 Discussion

We have shown how a demonstration can speed up the learning and how reinforcement learning then continues as the agent continues to explore. The planner further accelerates the learning. After the agent's initial fully guided trials it starts to run autonomously but is still not optimal. However, the learned policy did not benefit from demonstration.

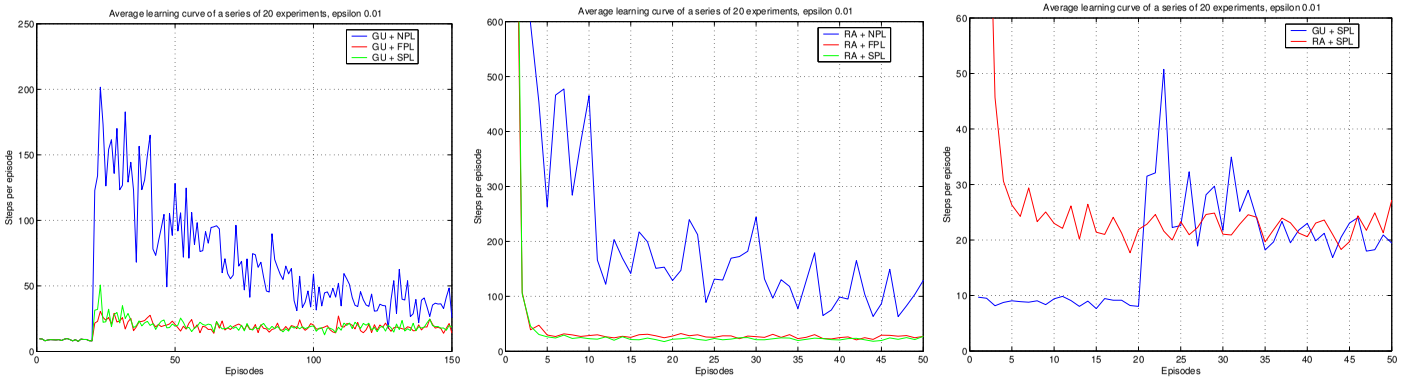


Figure 5.5: ***Left:*** *Guided (GU) Dyna-Q. The average number of steps per episodes measured over 20 experiments with 1331 states and 27 actions. Three planning methods are compared: non-planning (NPL), a full planner (FPL) and soft planner (SPL).* ***Middle:*** *Initially random (RA) Dyna-Q. The average number of steps per episodes measured over 20 experiments with 1331 states and 27 actions. Three planning methods is compared: non-planning (NPL), a full planner (FPL) and soft planner (SPL).* ***Right:*** *The guided case compared to the initially random case, the guided agent replays the demonstrated trajectories the first 20 steps, and is then released to work autonomously.*

5.3 Reinforcement Learning Using a Next-State-Planner

In the previous experiment the scenario is over-simplified, thus, not very realistic. The reason for the simplification is owed to an–otherwise large–state-action space, which would be unfeasible to use because of the computational limitations. To overcome these limitations the state space can be approximated, and the human demonstrations can be used as actions. Moreover, for imitation learning the robot needs a method by which it can transform observed actions into robot actions, not only by observation and interpretation, but also by executing the observed action and comparing the result to the demonstration. This implies a developmental structure where an action is first learned by observation and later modified to better fit the robot's morphology and the experience it gains upon executing the action.

The developmental learning approach was first proposed by Weng et al. [2001], as a way to have robots with developmental learning instead of pre-programmed knowledge. This approach emphasizes on the *learning* of skills, which are neither known at the time of programming, nor by the programmer. Furthermore, learning should be on-line and in real time and should first involve basic skills and then move to more complex skills.

We propose a three staged process for the robot to acquire novel skills from demonstration, where the stages are *observation*, *execution* and *self adaptation*. This will equip the robotic system with the above outlined developmental skills. In the experiments the robot first observes a demonstration, creates a model, and then executes its own version of the skill using the model. If the execution is successful a new model is created, based on the successfully executed motion.

Let us assume that planning and re-planning is performed in hand-state space. Our hypothesis is that in this case the self executed motions using models built from the robot's own motions are better adapted to the robot than a model built from the motions of the human demonstrator. This means that skills modeled form the robot's own actions should receive a higher reward than the skills directly modeled from human demonstrations. The purpose of the experiment is twofold: firstly, to test our hypothesis; secondly, to test our proposed reinforcement learning strategy for the fuzzy-modeling based next-state-planner, introduced in chapter 4.

Another important aspect of our method is that a skill which has been observed, modeled, and successfully executed can be stored for recognition purpose and reused later. This means that even if a particular demonstration would not be executable by the robot, an earlier demonstration–which resulted in an executable robot skill–can be recognized as an instance of this skill and used instead of the current demonstration.

5.3.1 Methodology

In the experiments in this section an imitation metric of a reaching trajectory are used for Q-learnig (section 5.1.2) of action selection. The major term in determining the success of the trajectory is the result of a grasp. In adding, the metric is based on the hand-state error (see section 4.1) and the minimum jerk (section 2.2.2). The hand-state error determines how much the motion resembles the demonstration, and the minimum jerk measure will penalize jerky motions. Here, the same next-state-planner as in section 4.2 is used for trajectory generation, but with a simpler method for determining the dynamics. The experiment will show how well a skill modeled from a demonstration performs, and how well a skill modeled from its own execution of this skill performs in comparison to each other. More importantly, the experiment will show how useful these skills are for arbitrary initial positions of the end-effector. Hence, we test the generalization capability of the planner and extend the results from the experiment presented in section 4.2.3.

The models obtained from human demonstration are used as actions, thus, the action space is discretized in form of models, i.e., skills. From our previous work (chapter 4), we know that theses actions can perform reaching motions sufficient enough for the robot to use as actions. This means that a full reaching motion–the trajectory–is used as an action, instead of applying action selection at each point in time to produce a trajectory.

The demonstration will be done in two stages: the first in which the environment is recorded, and a second where the task is shown. We use a simplistic modeling of the environment where all objects are modeled as box shaped objects. A more sophisticated modeling could be used if a better granularity of the workspace is needed [Charusta et al., 2009]. The learning process is illustrated in figure 5.6 containing five phases as follows:

Demonstration The data acquisition phase where the demonstrator performs the task. The hand state trajectory H_n^d is recorded.

Modeling The demonstrations are interpreted under the assumption that a pick-and-place task is performed. The pick and the place positions are determined, and the trajectories are modeled in hand-state space using fuzzy modeling $\hat{H}_n^d$.

Execution of Demonstrations After the demonstrations are interpreted and converted into hand-state trajectories, these trajectories are executed by the robot manipulator. The trajectory executed by the robot is denoted H_n^r.

Evaluation Trajectories executed by the manipulator are evaluated, and the ones that performed sufficiently well are remodeled. The evaluation criteria are based on how closely the hand-state trajectory is followed and if the grasps were successful.

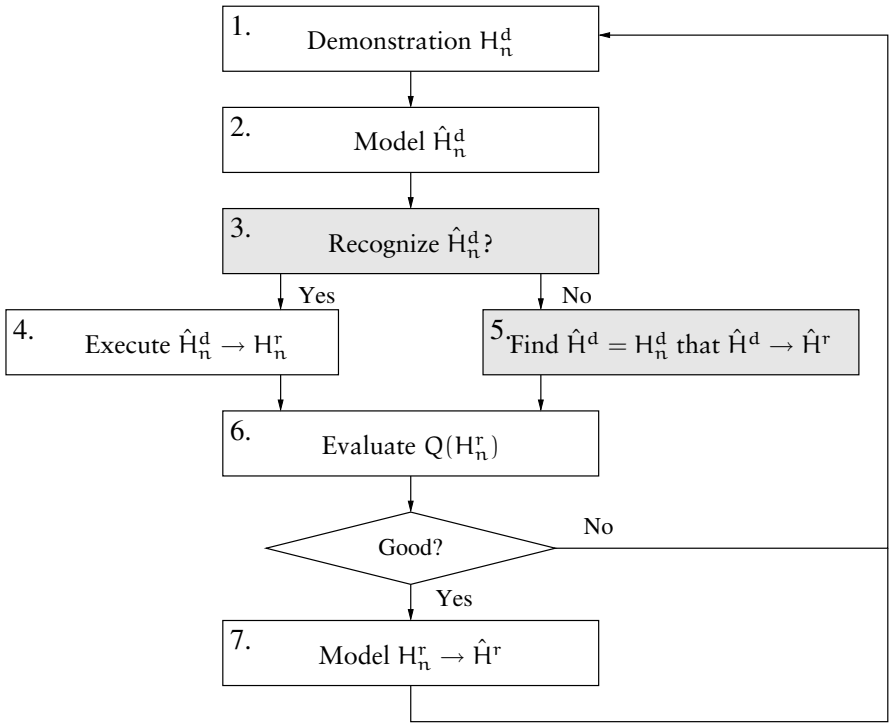


Figure 5.6: *The learning and evaluation process in our framework. The two gray-shaded boxes are not implemented for this experiment.*

Re-modeling If the evaluation shows that the executed motion performs well, the executed trajectory H_n^r is used to create a model $\hat{H}^r$ (exactly like the demonstration H_n^d). This means that a successful robot version of the demonstration is "worth" more that the human demonstration.

In figure 5.6 box number 3 referrers to the skill recognition, and is not implemented or tested in the experiment in this section. However, as new demonstrations H_n^d are observed and modeled these new models can be compared to previous models, e.g., H_1^d. In this way, skills can be recognized using the same scheme as described by Palm and Iliev [2006]. If a new demonstration (model) matches an already modeled trajectory, indicated by box number 5 in figure 5.6, the corresponding robot model H^r is used to execute the robot motion. If no match occurs (no recognition), the all the above described process are used to acquire the new skill.

For testing our hypothesis, the re-modeled trajectories are executed by the robot and evaluated using the same criteria as the initial evaluation. According to our hypothesis the re-modeled trajectories obtained from trajectories

executed by the manipulator, should perform better than the original models, obtained from human demonstrations. The hypothesis is tested using a reinforcement learning framework where models built both from demonstrated and self executed trajectories are used as actions. By using a reinforcement learning framework, an option for self improvements of actions is opened.

Generation of Robot Trajectories

For robot trajectory generation the next-state-planner described in section 4.2 is used. The reason for using that next-state-planner instead of the simplified version described in 4.3 is that we provide two types of demonstrations: one *task demonstration*, i.e., a pick-and-place of an object; one *environment demonstration*, i.e., the target objects in the workspace are shown to the robot. The environment demonstration provides an accurate description of the workspace for box shaped objects. A bounding box represents each object with position of the center and length, width and height, which are used to compute the orientation of the object. Since this description is more accurate than the object estimation from the task demonstration the next-state-planner described in section 4.2, which explicitly takes the goal into account, is better choice, compared to the planner described in section 4.3. The task demonstration contains the trajectories which the robot should execute to perform the task. Like in chapter 4 the notion of hand-state describes the object related trajectories. In these experiments the orientation interpolation between initial and final position is made using spherical linear interpolation [Shoemake, 1985].

One modification to the next-state-planner used in this section is a different weighting mechanism than the variance used in the experiment in section 4.3. Since the environment demonstration provides accurate data compared to the task demonstrations, the weight of the task demonstration is set ot 0 at the near the target object. An in reverse, the target object pose obtained from the environment demonstration receives the full weight of 1 at the end of the trajectory. Formally we express this as:

$$\left.\begin{array}{ll} \beta = & K(1-\Gamma(t)) \\ \gamma = & K\Gamma(t) \end{array}\right\} \; 0 \leqslant \Gamma \leqslant 1 \tag{5.7}$$

where β is the weight of the demonstration; γ is the weight of the target; and K is a fixed gain, in this experiment $K = 2$. The weighting function denoted $\Gamma(t)$ is dependent of the time:

$$\Gamma(t) = \left(\frac{t_{left}}{t_{final}}\right)^2 \tag{5.8}$$

where

$$t_{left} = \begin{cases} t_{final} - t & \text{IF} \quad t \leqslant t_{final} \\ 0 & \text{IF} \quad t > t_{final} \end{cases} \tag{5.9}$$

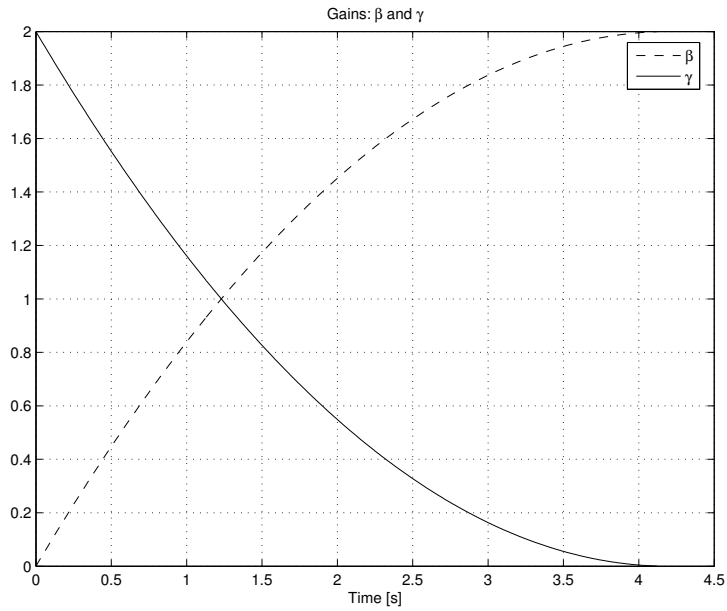


Figure 5.7: *The two gains β and γ in equation 5.10 during a reaching motion. β represents the weight of the target goal from the environment demonstration, and γ the weight of the task demonstration (the reaching motion).*

and t_{final} is the duration of the demonstration. This will lead the weight to shift from the demonstrated trajectory to the demonstrated target. Figure 5.7 shows β and γ as a function of time, when the reaching motion approaches the target object. The weight β represents the importance of the target pose recorded from the environment demonstration. Similarly, γ is the weight of the reaching motion in the task demonstration. During the motion, the importance shifts from the demonstrated trajectory to the demonstrated goal. A similar weighting mechanism was presented by Hersch et al. [2006], in an equivalent context. The dynamics of the NSP will be as in equation 4.16:

$$\ddot{H}(t) = \alpha(-\dot{H} + \beta(H_g - H) + \gamma(H_d(d) - H)) \tag{5.10}$$

where H_g is the hand-state goal, $H_d(d)$ the desired state at distance d, H is the current hand-state, $\dot{H}$, $\ddot{H}$ are the velocity and acceleration respectively, and α is a positive constant. In these experiments α was fixed at 8 and $dt = 0.01$s.

Evaluation using Q-learning

The actions of the robot have to be evaluated to enable the robot to improve its performance. The trajectory executed by the robot is evaluated based on three criteria:

- Deviation between the demonstrated and executed trajectories.
- Smoothness of the motion, less jerk is preferred.
- Successful or unsuccessful grasp.

Using a Q-learning framework (section 5.1.2), the reward function can be formulated as:

$$r = r_1 + r_2 + r_3 \tag{5.11}$$

where

$$r_1 = -\frac{1}{t_f} \sum_{t=0}^{t=final} |H^r(t) - H(t)| \tag{5.12}$$

$$r_2 = -\frac{1}{t_f} \sum_{t=0}^{t=final} \dddot{x}^2(t) \tag{5.13}$$

$$r_3 = \begin{cases} -100 & \text{if Failure} \\ 0 & \text{if Success, but overshoot} \\ +100 & \text{if Success} \end{cases} \tag{5.14}$$

where $H^r(t)$ is the hand-state trajectory of the robot, $H(t)$ is the hand-state of the demonstration. The second term of the reward function is proportional to the jerk of the motion, where t_f is the duration of the motion, t_0 is the staring time and $\dddot{x}$ is the third derivative of the motion. For the third term, "failure" means a failed grasp and "success" means that the box shaped object was successfully grasped. There is another case where the end-effector performs a successful grasp but with a slight overshoot. An overshoot means that the target is sightly missed, but the end-effector then returns to the target. Overshooting is and unwanted property, since it might displace the target, or even worse damage the gripper or target. This case is considered as neither a complete "failure" or "success" and received zero reward.

When the robot has executed the trajectories and received the subsequent rewards and computed the accumulated rewards, it determines what models to employ. The actions that received a positive reward are re-modeled, but this time as robot trajectories using the hand-state trajectory of the robot. This will give less discrepancies between the modeled and the executed trajectory,

Parameter	Value	Description
Diag. only	1	1/0 to update only the diagonal distance metric
Penalty	10^{-7}	A smoothness bias
Kernel	Gaussian	Gaussian or BiSquare
D_{init}	$eye(dim) * 25$	Initial distance weighting matrix
α_{init}	$ones(dim) * 100$	Initial learning rates
w_{gen}	0.2	Threshold to create a new receptive field
Meta	1	1/0 allow the use of a meta learning parameter
$Meta_{rate}$	100	The meta learning rate
λ_{init}	0.995	Initial forgetting rate
λ_{final}	0.9999	Final forgetting rate

Table 5.4: *The Locally Weighted Projection Regression parameters and their corresponding values used in our experiment. Note that these parameter names should not be confused with the same names used in other methods, for example the learning rate in LWPR named* α.

thus resulting in a higher reward. In Q-learning a value, a quality measure, is assigned for each state-action pair by the rule:

$$Q(s, a) = Q(s, a) + \alpha * r \tag{5.15}$$

where s is the state, in our case the joint angles of the manipulator, a are the actions, i.e., each model from the demonstration, and α is a step size parameter (not to confuse with the dynamic parameter α in Egn. 5.10). The reason for using the joint space as the state space is the highly non-linear relationship between joint space and Cartesian- or hand-state space: two neighboring points in joint space are neighboring in Cartesian space but not the other way around (see Appendix A.1.3 for more details). This means that action selection is better made in joint space since the same action is more likely to be suitable for two neighboring points than in Cartesian- or hand-state space.

Unlike most other applications of reinforcement learning, there is only one state action transition, meaning that from a given position only one action is executed and then judged upon.

To approximate the Q-function we used Locally Weighted Projection Regression (LWRP)[1] as suggested by Vijayakumar et al. [2005], see their paper for details. Table 5.4 summarizes the parameters of LWRP used in our experiment.

5.3.2 Experimental Results

The motion capturing system we used to record the demonstrations is the Impulse system from PhaseSpace, described in section 4.3.1, with the addition of tactile sensors, i.e., force sensing resistors. The sensors are mounted on the fingertips of the glove, shown in figure 5.8. We define a grasp as when contact is

[1] Available at: http://www-clmc.usc.edu/software/lwpr

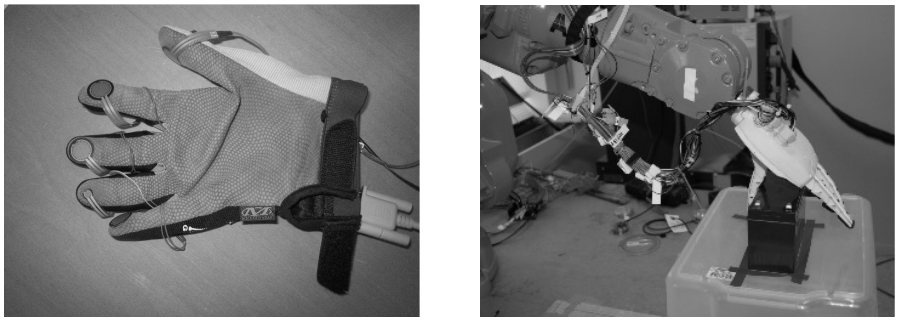


Figure 5.8: ***Left:*** *The glove in the motion capturing system with the tactile sensors mounted on each finger tip.* ***Right:*** *The anthropomorphic gripper KTHand used in the second experiment.*

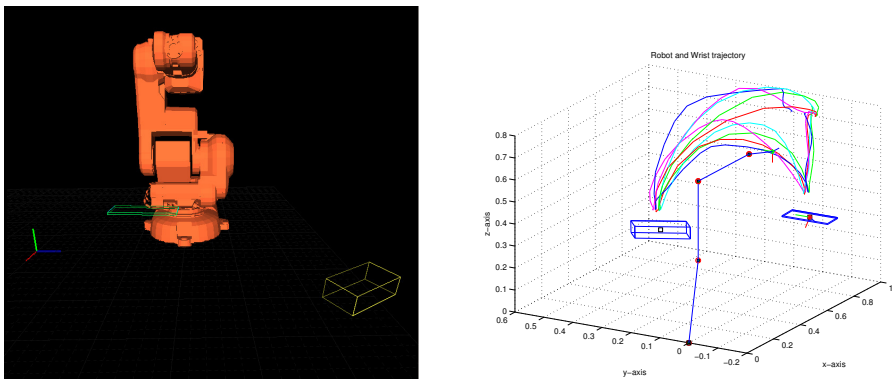


Figure 5.9: ***Left:*** *The result of an environment demonstration.* ***Right:*** *Five sample demonstrated trajectories (the wrist is plotted), the robot and the two objets of interest. The object to the right is marked with a red star to indicate that this object was the nearest when a grasp was detected in the task demonstration.*

detected at the thumb sensor and one additional finger. This means that only grasps which include the thumb and one other finger can be detected. No grasp recognition is necessary since only one grasp type is possible on the grippers we used in this experiment. When a grasp is detected the distance to each object in the workspace is measured and the nearest object, if below some distance threshold, is identified as the target object. The robot used in this experiment is the 6 DOF serial manipulator ABB IRB140 described in section 3.3.5.

The demonstrations were performed with the teacher standing in front of the robot and performing the task demonstrations in two stages. First, the environment is demonstrated by tactile exploration of the workspace. The demonstrator touches the objects of interest; with special care so that the boundaries of each object are correctly captured. Second, the task is demonstrated where

the teacher starts with the hand in a similar position as the end-effector of the robot would be in its home position, i.e., $\Theta = [0\ 0\ 0\ 0\ 0\ 0]^T$. The result from the environment and task demonstrations is shown in figure 5.9, with the base frame of the robot as the reference frame. The object is determined from the environment demonstration, since it is more exact compared to the task demonstration.

A Small Parallel Gripper–Failed Grasps

The gripper used for this experiment is a pneumatic parallel gripper with a grasping width of 8 mm, where the distance between the jaws is $42 - 50$ mm. The base of the object used for the task is 43×59 mm. Given a perfect alignment and orientation, the required accuracy of the positioning of the gripper is $(50 - 43)/2 = 3.5$ mm in the O-direction of the gripper (the jaws), which is the direction which requires the highest precision. A series of pictures in figure 5.11 shows different gripper orientations around each axis, given a perfect positioning and alignment around the other axis. The real required accuracy is much higher since the different measures are strongly correlated. This means that the highest tolerance, 3.5 mm, will be further lowered by the required accuracy from the other axes, rendering the real value less than 3.5 mm.

The experiment was carried out as follows. Five task demonstrations were recorded and one workspace demonstration was recorded. Then in the modeling phase the recorded trajectories were modeled using fuzzy modeling. One of the models and the generated trajectory is shown in figure 5.10. Each of the generated trajectories was then executed on the real manipulator. However, none of these generated reaching trajectories were successful: all failed in the final segment of the trajectory, as shown in figure 5.12, where the gripper collides with the object.

The reasons for the failure are multiple: The size of the gripper is small compared to the object to be grasped. A human demonstration is not very exact and when the subject demonstrates a pick-and-place task the focus is not on precision. Clearly, under our assumptions we have reached the limit to where this approach can be used. One solution which can remedy the inaccuracy of the human task demonstration is to associate an approach motion towards the object obtained from the more accurate environment demonstrations. An approach vector would also depend on the type of gripper for grasping.

The Anthropomorphic Gripper–Successful Grasping

In this experiment we use the anthropomorphic gripper KTHand, which can perform power grasps (i.e., cylindrical and spherical grasps) using a hybrid position/force controller. For details on the KTHand, see Tegin et al. [2008].

Twenty demonstrations of the same pick-and-place task were performed, where only the first reaching part of the task was used to train a reaching skill.

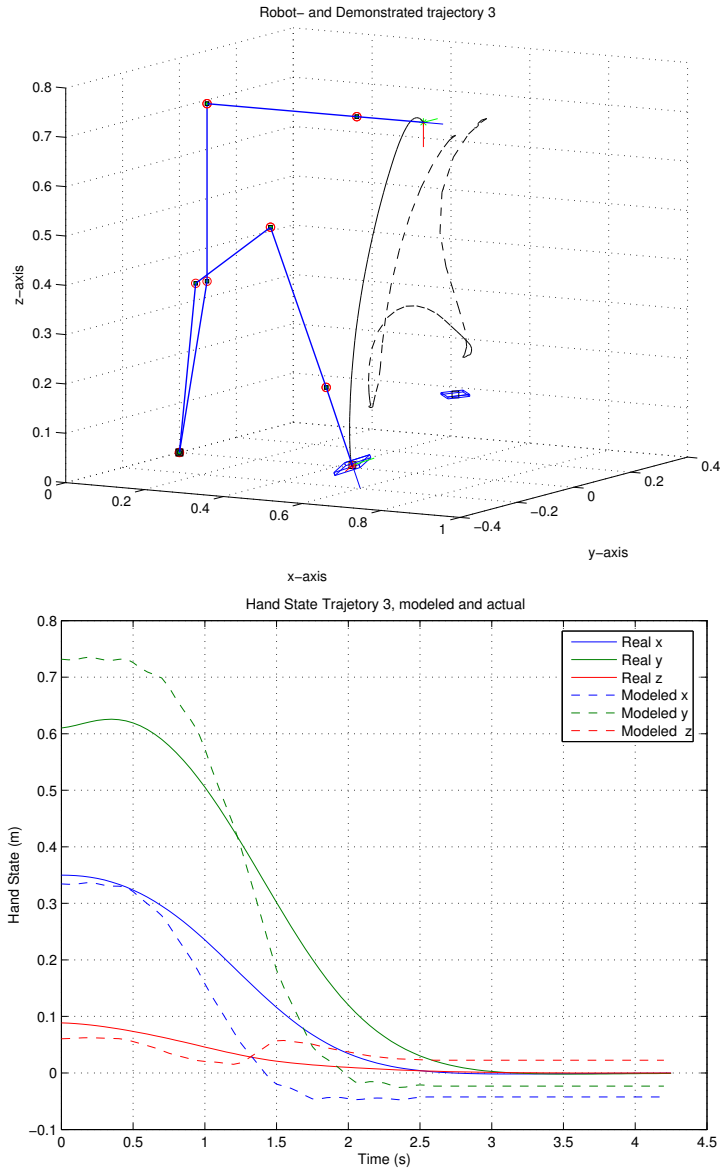


Figure 5.10: *The top graph shows the human demonstration (wrist) and the trajectory generated by the robot with the parallel gripper. The bottom graph shows the position in time for* x,y *and* z *both the modeled (from demonstration) and the executed (using the planner).*

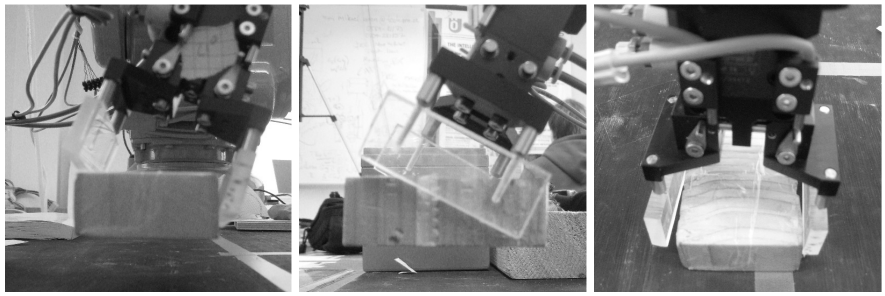


Figure 5.11: *Pose of the gripper to show how much it can rotate around each axis, given that the positioning and orientation around the other axis is exact.* ***Left:*** *Rotation around the* N*-vector of the gripper.* ***Middle:*** *Rotation around the* O*-vector of the gripper.* ***Right:*** *Rotation around the* A*-vector of the gripper.*

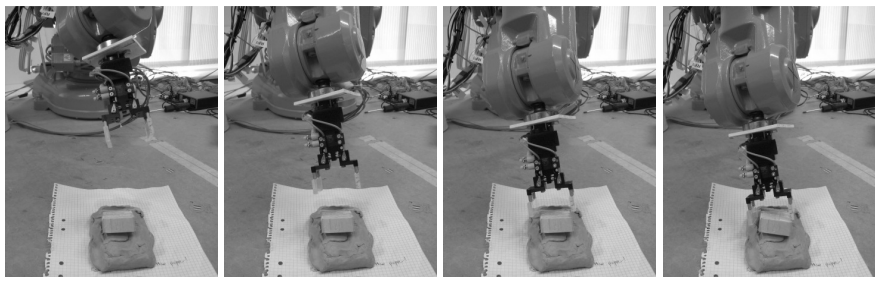


Figure 5.12: *The parallel gripper fails to grasp the box shaped object. The object is positioned on clay so that there is no assumption on its orientation, besides that it can be grasped.*

Five of these demonstrations are shown in figure 5.9. All demonstrations were modeled using fuzzy time-modeling, where the position of the index finger and orientation of the wrist were used to record the trajectory. Three of the modeled trajectories are shown in figures 5.13 to 5.15 in the top graph as dashed lines. The reason for using the index finger instead of–what would be more appropriate–the center point between the index finger and the tip of the thumb is that the LED on the thumb was often hidden resulting from occlusions during the motion. To compute the orientation of the wrist, three LEDs must be visible during the motion. The back of the hand is the best choice since three LEDs are mounted there and they are most of the time visible for at least three cameras. Thus, the number of occlusions is minimized.

All models were then executed by the robot to test the performance of each, evaluated using the criteria in equation 5.11. Three of the executed trajectories are shown in figures 5.13 to 5.15 in the top graph as solid lines and in the bottom graph as Cartesian trajectories. At the end of the demonstrations in figures 5.13 to 5.15 the difference between the model and the actual trajectory is up to 8 cm (in y-direction). Since the modeled is created from the trajectory

of the fingertip of the index finger, the end of the trajectory will be displaced from the center of the target object. This is due to the fact that the index finger will be at the side of the box shaped object during the grasp. Recall from the definition of the hand-state space (section 4.1) that the target object is the frame in which all motions are in relation to. This mean that the origin of the hand-state is the top center of the target object. The point attractor of equation 5.10 for the trajectory generation is switch from the task demonstration to the target object, as described by equation 5.7. Therefore, the point $[0, 0, 0]$ becomes the point attractor at the end of the motion.

The hand-state error and minimum jerk can be evaluated already in the generation state in simulation, i.e., before the actual execution on the real robot. The grasp success is evaluated from real execution. In the real execution the robot starts in the home position $\Theta = [0\ 0\ 0\ 0\ 0\ 0]^T$ deg, with a small perturbation added on the last three joints since the home position is a singular configuration. All models succeeded in generating trajectory which positioned the end-effector in such a way that a successful grasp can be performed, resulting in a positive reward of $+100$. This reward is then decreased by adding a negative reward from hand-state error and jerk, as described by equations 5.11-5.14. In table 5.5 this is shown in the rows with joint configuration $\Theta = [0\ 0\ 0\ 0\ 0\ 0]^T$ deg. The models from human demonstration H_6 and H_{11} performed best when the robot executed them. Hence, these were selected to be remodeled into robot skills: R_1 and R_2, meaning that the robot trajectory is used to create the fuzzy models. Worst performance was obtained in H_{19} and H_{20}, where H_{19} received a negative Q-value. Finally, R_1 and R_2 were tested from the home configuration and evaluated using the same criteria, shown in figure 5.16 and 5.17. As expected, their performance was better than all others, since the hand-state error is reduced.

To evaluate if these new skills also performed better than the skills modeled from human demonstration, H_6 and H_{11} were selected for comparison, since they performed best. In addition we also selected the worst model H_{20} of the remaining models, with a positive Q-value, thus, excluding H_{19} since a negative reward might mean a failed grasp. Twelve joint configurations were chosen approximately corresponding to 1/4 of the full joint range at each joint, except last joint where 1/8 of the range was used. Some modifications were also done resulting from workspace restrictions, i.e., $\Theta = [0\ 20\ 40\ 1\ 1\ 1]^T$ deg. The resulting Q-values form these joint configurations can be seen in table 5.5, with the highest Q-values highlighted. From joint configurations $\Theta = [0\ -45\ 0\ 1\ 1\ 1]^T$ deg and $\Theta = [0\ 0\ 0\ 1\ -55\ 1]^T$ deg (dark rows in table 5.5) non of the models where able to execute a successful reaching motion. This mean that new demonstrations should be done from these positions. Neither H_6 nor H_{11} performed best in any test case, however the worst model H_{20} was the only one to perform a successful reaching motion from configuration $\Theta = [0\ 0\ -50\ 1\ 1\ 1]^T$ deg, resulting in the highest Q-value. From the rest of the configurations robot skill R_1 or R_2 performed best, thus confirming

	Q-values										
	Human Actions										
Joint configuration, $\mathbf{q}$	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_9	H_{10}	H_{11}
$[0\ 0\ 0\ 0\ 0\ 0]^T$	0.7572	0.7854	0.8120	0.8596	0.6864	0.9521	0.7410	0.8966	0.7738	0.8659	0.9964
$[-45\ 0\ 0\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	1.6079	N/A	N/A	N/A	N/A	1.7029
$[0\ -45\ 0\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	−0.3890	N/A	N/A	N/A	N/A	−0.2960
$[0\ 0\ -50\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	−0.3895	N/A	N/A	N/A	N/A	−0.2968
$[0\ 0\ 0\ -50\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	1.6104	N/A	N/A	N/A	N/A	1.7032
$[0\ 0\ 0\ 1\ -55\ 1]^T$	N/A	N/A	N/A	N/A	N/A	−0.3895	N/A	N/A	N/A	N/A	−0.2968
$[0\ 0\ 0\ 1\ 1\ -50]^T$	N/A	N/A	N/A	N/A	N/A	1.6102	N/A	N/A	N/A	N/A	1.7031
$[45\ 0\ 0\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	1.6102	N/A	N/A	N/A	N/A	1.7030
$[0\ 20\ 40\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	1.6104	N/A	N/A	N/A	N/A	1.7032
$[0\ 0\ 25\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	1.5848	N/A	N/A	N/A	N/A	1.6515
$[0\ 0\ 0\ 50\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	1.6105	N/A	N/A	N/A	N/A	1.7031
$[0\ 0\ 0\ 1\ 55\ 1]^T$	N/A	N/A	N/A	N/A	N/A	1.6105	N/A	N/A	N/A	N/A	1.7032
$[0\ 0\ 0\ 1\ 1\ 50]^T$	N/A	N/A	N/A	N/A	N/A	1.6105	N/A	N/A	N/A	N/A	1.7032

	Human Actions									Robot Actions	
Joint configuration, $\mathbf{q}$	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{17}	H_{18}	H_{19}	H_{20}	R_1	R_2
$[0\ 0\ 0\ 0\ 0\ 0]^T$	0.8555	0.8349	0.8334	0.8319	0.8481	0.7656	0.7572	−1.9459	0.6745	1.0769	1.0437
$[-45\ 0\ 0\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1207	−0.0670	1.7847
$[0\ -45\ 0\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	−0.8784	−0.0670	−0.2143
$[0\ 0\ -50\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1208	−0.0676	−0.2150
$[0\ 0\ 0\ -50\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1208	−0.0675	1.7850
$[0\ 0\ 0\ 1\ -55\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	−0.8792	−0.0676	−0.2150
$[0\ 0\ 0\ 1\ 1\ -50]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1208	−0.0675	1.7850
$[45\ 0\ 0\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1206	1.9320	1.7847
$[0\ 20\ 40\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1207	1.9324	1.7850
$[0\ 0\ 25\ 1\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.0994	1.9017	1.7275
$[0\ 0\ 0\ 50\ 1\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1207	1.9324	1.7850
$[0\ 0\ 0\ 1\ 55\ 1]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1208	1.9324	1.7850
$[0\ 0\ 0\ 1\ 1\ 50]^T$	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1.1208	1.9324	1.7850

Table 5.5: *The Q-values for selected joint configurations. When the robot imitated the human actions* H_6 *and* H_{11} *(light gray) from its home configuration, it received the highest Q-value and used these two to create the two new action models* R_1 *and* R_2*. Twelve other configurations were selected to test how well these modeled action performed. To compare the two original motions* H_6 *and* H_{11} *were selected since they performed best and* H_{20} *which had the lowest positive Q-value.*

our hypothesis that skills built from own experience are better adapted than skills directly modeled from observation. Figure 5.18 show four sample configurations where the robot skill R_1 and R_2 are used to generate actions from configurations different from the trained position. In figure 5.19 two sequences are shown where the robot executes the reaching skills from from the trained position, and from the tested position.

5.3.3 Discussion

Human demonstrations have been shown to provide knowledge to produce models good enough for the robot to use as its own skills. The robot gains experience from human demonstration, and we have shown how the robot can improve upon this when the self executed motions are performed. Thus, we apply reinforcement learning on each skill model–constructed from demonstration– as an action to learn a Q-function for each state to select (or suppress) different actions. The reinforcement learning process provides a developmental structure to our system, where the robot can observe, imitate, and improve. It also enables the robot to generalize skills so that one skill can be used for different tasks, and determines when to select what skill. Furthermore, if the learning process is implemented as a continuing process it enables the robot to improve performance not only at the initial stage, as we tested, but over a longer time scale. The method relies on an external teacher to judge if the reaching-skill was successful. The ability to forget a poorly performed skill or a skill that produced a very similar trajectory compared to other skills will also be part of our future work. A pruning ability will prevent the database of skills from becoming too large.

In the experiments, we also saw the limits of our approach. Grippers with low tolerance requires both an accurate environment demonstration as well as an accurate approach to the object. The former was provided by the environment demonstration, but the latter could not be acquired from the task demonstration. However, for autonomous grippers, such as the KTHand, the method provides effective skill learning with good generalization capability.

Since our modeling method also is suitable for recognition [Palm and Iliev, 2006, Ju et al., 2008], future work will include recognition of skills to allow the robot not only to imitate, but instead use its own experience to execute the demonstrated skill in its own way. Future experiments will test how the performance will change if more demonstrations of the same task, but with a greater variety, are provided to the robot. Of interest if how many demonstrations are needed before the robot have reached sufficient performance.

5.4 Summary

We have shown how demonstrations of reaching motions can speed up a slow learning process. Furthermore, we have shown how an imitation metric can

be used to evaluate and select actions among a demonstrated set of skills in a reinforcement learning framework. In section 5.2, the state space was the Cartesian space and the actions-space was a step in a discrete direction. The state space was also very coarse, which is not suitable in a real application.

In section 5.3 the reward concept from reinforcement learning was used as a metric of imitation. The joint angles represented the states, and the models from human demonstrations were the actions, meaning that one action will bring the end-effector to the goal in one single action. The fuzzy modeling based next-state-planner performed the actions, which combined with the imitation metric provided a developmental approach where the robot can improve its performance after own experience of a task.

In our first approach a discrete representation of the state-action space was used, which renders the approach over-simplified. In the second approach the state space was approximated using LWPR to have a continuous representation, thus, a realistic scenario. Furthermore, the actions were represented by skills already known to perform well from human demonstration, so we still have a discrete representation of the action space. This leads to an approach which is practically possible to implement with good performance results in a real reach to grasp scenario. Compared to the first approach, which produces new skills by performing action selection in every step along the trajectory, in the second approach new actions can be created based on the actual execution of a skill, i.e., the full trajectory. It is impractical to have a continuos state-action space with no initial knowledge of the skill to learn since the search space will be enormous. Our alternative have showed that a practical implementation with initial knowledge is realistic and provides good results and the possibility to improve the performance.

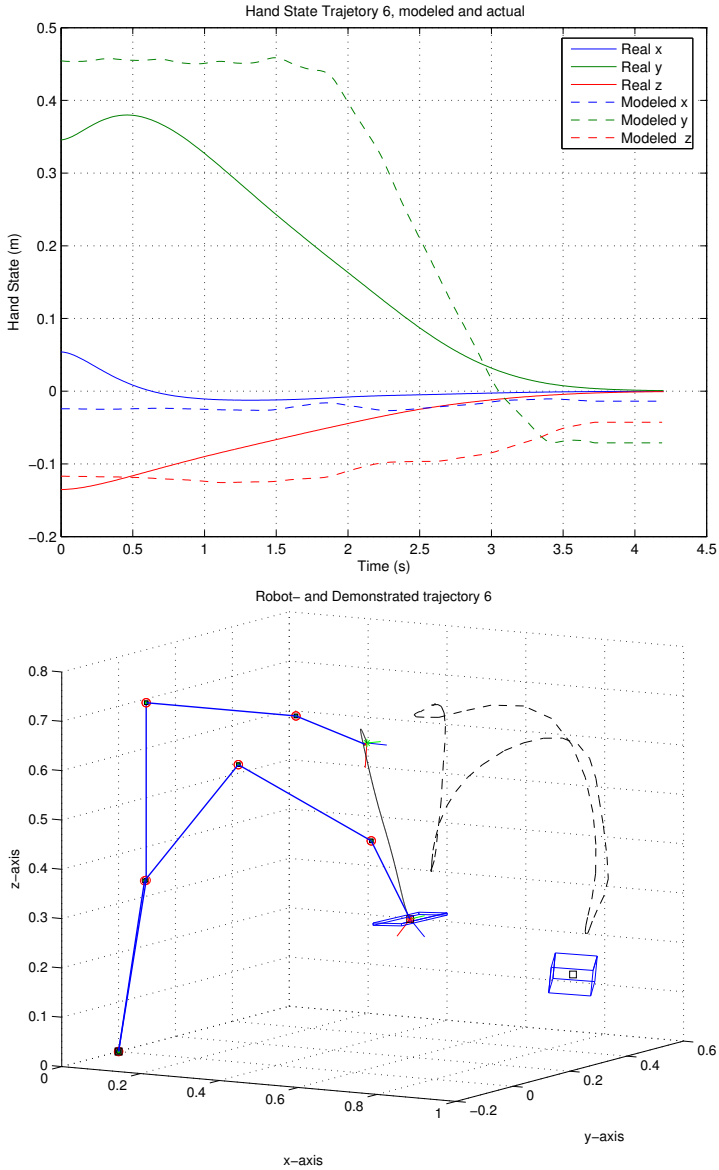


Figure 5.13: *Demonstration number* H_6*, one of the two best performing models, and the generated reaching trajectory the robot executes.*

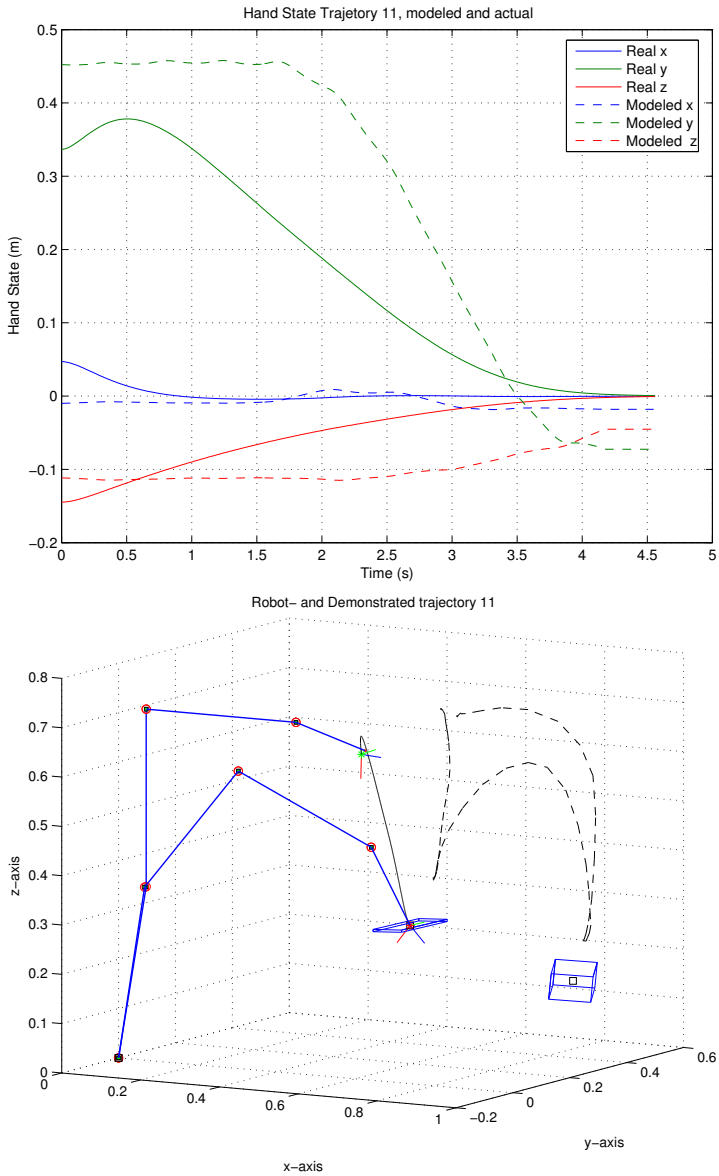


Figure 5.14: *Demonstration number* H_{11}*, one of the two best performing models, and the generated reaching trajectory the robot executes.*

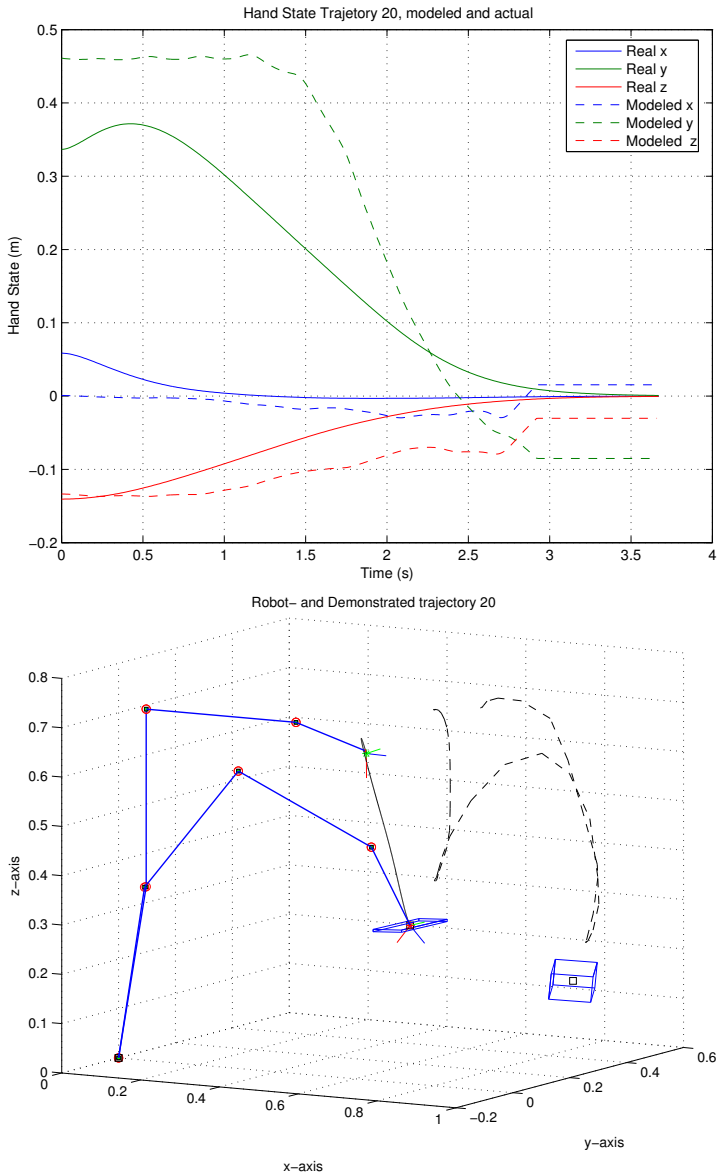


Figure 5.15: *Demonstration number* H_{20}*, the worst performing model with a positive Q-valus, and the generated reaching trajectory the robot executes.*

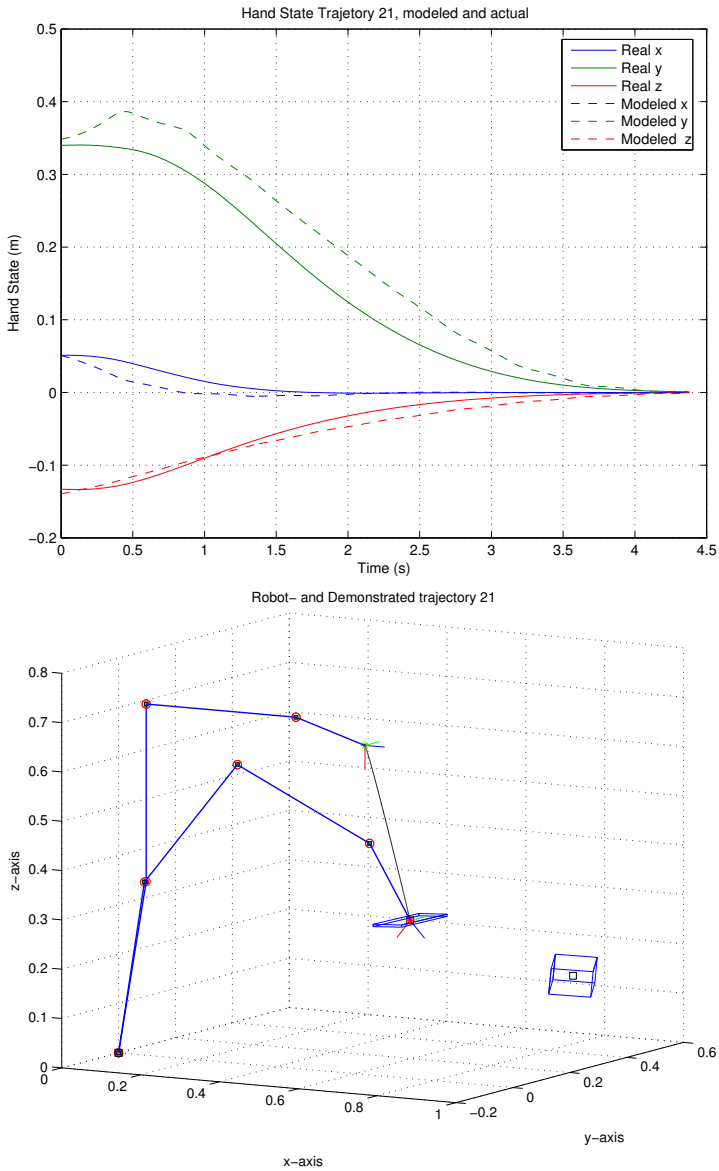


Figure 5.16: *Robot model* R_1 *and the generated reaching trajectory the robot executes.*

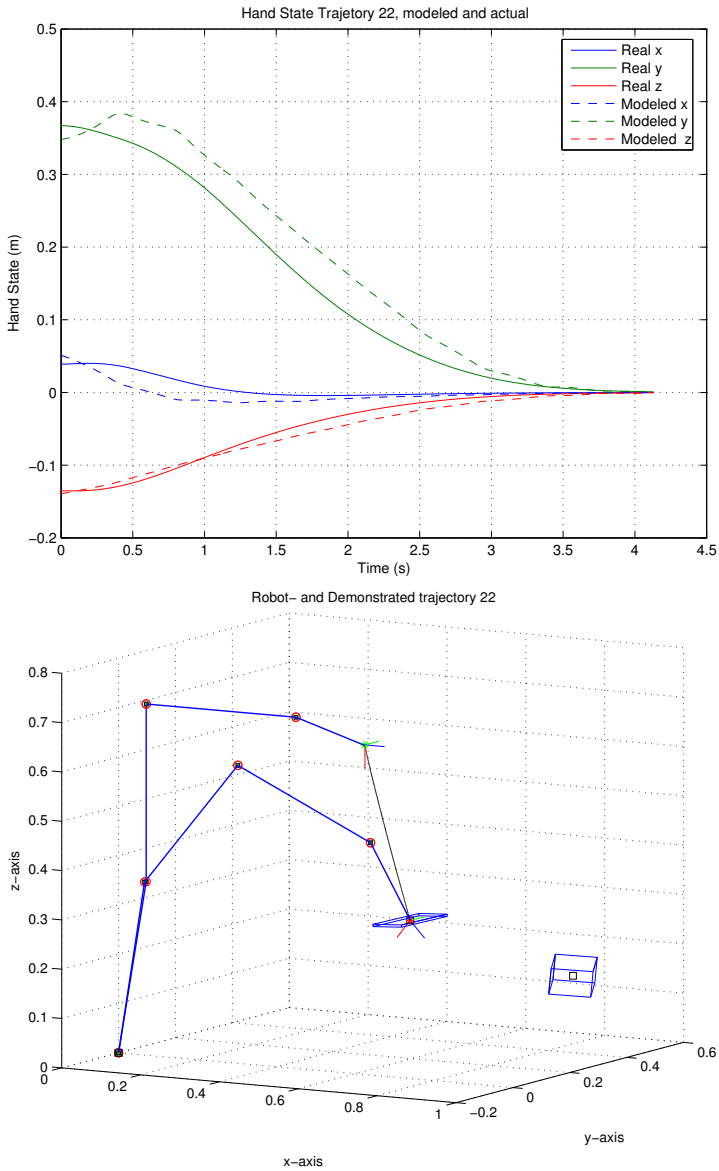


Figure 5.17: *Robot model* R_2 *and the generated reaching trajectory the robot executes.*

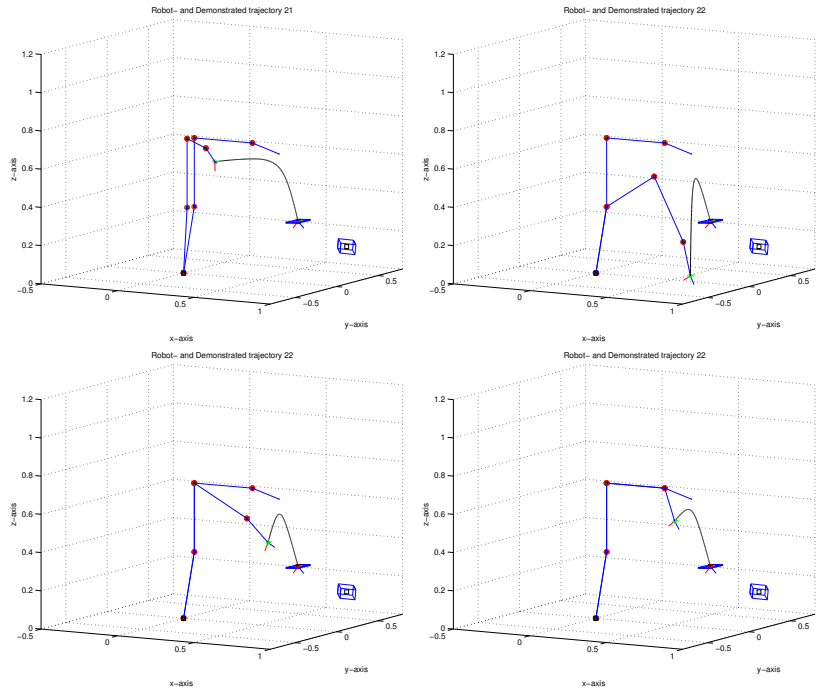
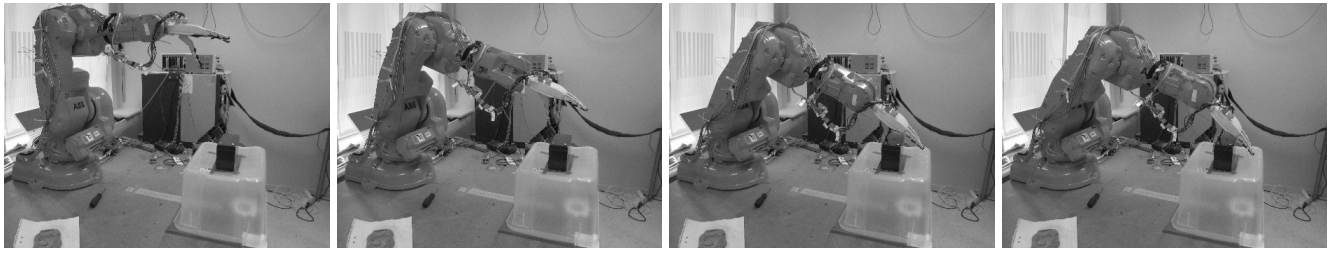


Figure 5.18: *Reaching from different initial positions. Four sample configurations,* $\Theta = [-45\ 0\ 0\ 1\ 1\ 1]^T$, $\Theta = [0\ 20\ 40\ 1\ 1\ 1]^T$, $\Theta = [0\ 0\ 25\ 1\ 1\ 1]^T$ *and* $\Theta = [0\ 0\ 0\ 1\ 55\ 1]^T$.

R_2 from configuration $\Theta = [0\ 0\ 0\ 0\ 0\ 0]^T$



R_2 from configuration $\Theta = [0\ 20\ 40\ 1\ 1\ 1]^T$

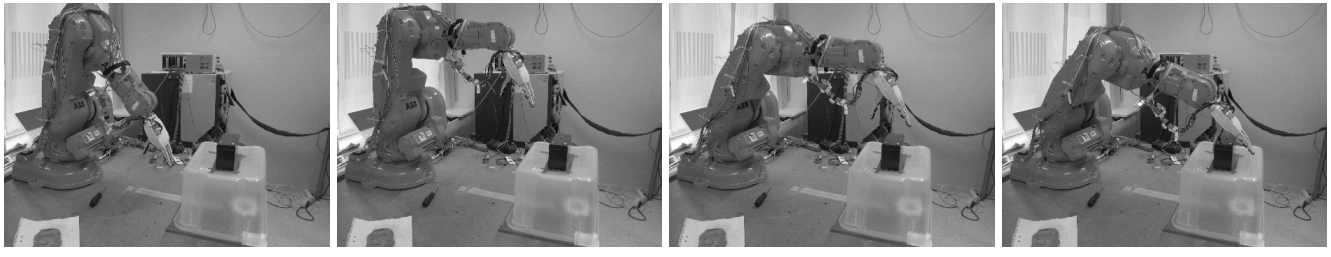


Figure 5.19: *The top series show the robot grasping the box shaped object from configuration* $\Theta = [0\ 0\ 0\ 0\ 0\ 0]^T$ *using robot skill* R_2 . *The bottom graph show the same skill* R_2 *but from a different initial position than the trained one:* $\Theta = [0\ 20\ 40\ 1\ 1\ 1]^T$.

Chapter 6
Conclusions

This chapter first summarizes and discusses the work, the main contributions, and the experimental results of the thesis. An important part of the discussion is the lessons learned during the study, and how these were addressed. Secondly, a discussion follows which points out the strengths as well as the weaknesses, of the approach in this thesis. Finally, building on what we learned, we provide some future research directions, addressing the weaknesses in the approach and hypothesizing on how to further develop the system.

6.1 Summary and Discussion

The scope of this thesis was to investigate how to–in the context of robot programming–interpret and use human demonstrations and to infer skill knowledge and transfer this knowledge to the robot. As a result, we have developed a method for Programming-by-Demonstration (PbD) for reaching and grasping tasks.

A large part of this project was to create the infrastructure needed for PbD. This work includes developing a system architecture containing a manipulator with a suitable motion capturing system, and providing the software around this platform. For the choices of motion capturing system, we have learned some important lessons in this process. Systems that measure position relative to a base attached to the user which is also moving are hard to use for teaching of object manipulation and goal directed motions. Accuracy is also a crucial point since object manipulation typically requires *at least* centimeter precision. If object interaction is not a requirement motion capturing systems like ShapeTape are applicable, for example, when recording human motions for animation purposes. The magnetic tracker used to track the wrist or index finger position also had its shortcoming in that application. It is not possible to use the tracker in the near vicinity of the robot since large metal parts distort the magnetic field. Hence, if the robot and the user should share the same workspace, this type of sensor is not an option. Or final choice was a marker

based motion capturing system using stereo cameras. It is accurate enough but suffers from occlusions, where the ShapeTape sensor and the magnetic tracker have the advantage.

A true vision system, with no constrains on the user, would be more convenient and a time consuming calibration process would be unnecessary. Furthermore, such a system would enable the robot to explore the environment, its own actions, and the effect the actions have, something which is not possible with our current system since it requires markers. The approach presented in this thesis is complemented by such a developmental process where the hand-state space can evolve from exploration.

To use grippers with sensing or built in autonomy alleviates the accuracy problem for positioning. This is illustrated by the experiments with a vacuum gripper with contact sensor, and the hybrid force/position controlled anthropomorphic hand. In both cases, the success in grasping an object is dependent on the autonomy of the gripper, which in turn is dependent on an accurate enough (defined for each gripper) reaching motion to execute a successful grasp.

The fundamental correspondence problem–the fact that embodied agents generally have different morphologies–in imitation learning have also been addressed in this thesis. In our first experiment, section 3.2, we investigated how to learn a model-free mapping from human to a robot arm, based on a reversed imitation scheme where the human imitates the robot. However, the performance was not satisfactory because the trained algorithm produces a position error which often prevents a sufficiently accurate placement of the gripper. To improve the modeling of the human motions we introduced a new way of encoding demonstrations–fuzzy-modeling using time-clustering. The fuzzy modeling method uses time as model input on a predefined number of cluster centers. The method shows excellent modeling performance and preserves the dynamical (humanlike) properties of the motion.

To allow the robot to interpret the human motions as its own, we employ a hand-state space representation as a common basis between the human and the robot. For the robot, to adjust its path towards an object a class of methods called next-state-planners can be used. Next-state-planners are biologically inspired by human motions and adapts the trajectory while the motion is executed. We contributed to the design of a next-state-planner, which includes the advantages of fuzzy modeling and executes the motion in hand-state space. To determine the importance of different parts of the trajectory the variance of the demonstrated trajectories is computed as a function of the distance to the target. This variance determines the gain for the planner, to follow the trajectory closely for parts of the trajectory with low variance.

One disadvantage of our approach regards the fact that it operates in Cartesian space (hand-state space) which may lead to unreachable joint space trajectories. Although it is possible to avoid unreachable joint configurations [Hersch and Billard, 2006], this will lead to a trajectory which will *not* follow the demonstration. Either the robot performs the task in a way that is possible

because of joint constraints but not as demonstrated, or it can ask for more information from the teacher.

In a simple environment where the task is known and a set of predefined skills is available, PbD can be applied with some effort. This was demonstrated in section 3.3. However, the generalization capabilities of such a method are limited and will require new demonstrations even for a slightly modified task. To generalize the learned skill the next-state-planner was used for trajectory generation from arbitrary positions in the experiments in chapter 4. We have shown that this method can generate executable robot trajectories based on current and past human demonstrations despite morphological differences. To illustrate the trajectory planner's ability to generalize, several experiments were performed, using an industrial robot arm that executes various reaching motions and performs power grasping with a three-fingered hand.

We have also discussed the possibility to learn reaching motions by using reinforcement learning. Our experiment showed that providing a demonstration speeds up Dyna-Q learning, but the environment was simplistic and would require a function approximation of the state space to make a realistic scenario. However, reinforcement learning can also be applied on a different level where it can decide what action (i.e., skill) to use in different points in hand-state space. We applied reinforcement learning in hand-state space and used fuzzy models–constructed from demonstration–as actions to learn a Q-function for each state to select (or suppress) models. In this way, the demonstrated models which perform well on the robot are re-modelled as robot actions. Then, these actions are incorporated into the robot's own movement repertoire and thereby increased the number of skills which the robot can perform.

The experiments in this thesis only concern reaching motions and pick-and-place tasks. Reaching motions are one of the most basic skills, thus, it is important that the robot can learn them for different types of objects, in different task-contexts and can generalize this knowledge. If we exclude object manipulation, such as clicking a button or flipping through the pages of a book, the pick-and-place task is probably the most common task to perform in our daily activities. To perform other tasks such as stacking objects or insert an object into another requires pick-and-place. For future work it is important to enable the system to learn other types of tasks. To include a wider spectrum of skills, future work should investigate rhythmic motions, since they are different in nature from the discrete reaching motions. Another shortcoming is the absence of obstacle avoidance ability, which we discuss in the future work section next.

The main contributions in this thesis can be summarized as follows:

1. A *model free mapping* from human positions to robot positioning. The advantage is to *learn* the mapping, instead of defining it.

2. Fuzzy modeling for encoding a skill, as a function of time: *time clustering*, and distance: *distance clustering*. Fuzzy modeling provides simple

and compact trajectory encoding with the advantage of preserving of dynamical properties.

3. A *next-state-planner* execute the time clustered trajectories. The planner handles the differences in configuration in relation to the object and morphological differences, thus contributes in the handling of the correspondence problem.

4. The notion of *hand-states* have been applied to PbD, where it provides the benefit of easy synchronization of reaching with grasping.

5. By using skills as actions in a reinforcement learning framework, the learning is made practical since the search space is reduced with a limited set of good performing actions. This provides a major advantage over classical reinforcement learning.

6.2 Future Work

Currently, artificial learning-by-imitation systems are restricted to very specific tasks. Using the task learning approach (see section 2.4) the demonstration becomes very hard to interpret when it violates the assumptions. Analogously, in the trajectory learning approach (see section 2.4) the trajectory must be generalized, stored and most importantly–put into context of a task. To link or merge these two concepts is a topic for future work.

On a smaller scale, the future work can be separated into three categories: *Sensing*, *Execution of Grasptype* and *Learning*.

Sensing Incorporating the robot's own perception into the loop will enable the robot to learn from its own experience. This direction for future research will include perception in the learning process, for example, to recognize and track objects, and to identify their affordances. In this way, the robot would not need to rely blindly on human demonstrations and would be able to work in non-static environments.

Execution of Grasptype To test the hand-state approach on different types of objects–with different affordances–will provide knowledge on how well the approach scale to different grasp types. Of special interest are precision grasps where high accuracy is needed.

Learning More experimental data will test our hypothesis on using the Euclidean distance to the object as a state variable; e.g., circular motions will need additional variables. Furthermore, the variable that determines the dynamics of the next-state-planner should be *learned* from experimental data, instead of predefined, to be able to generate trajectories.

Appendix A
Appendix A: Robotics Refresher

This chapter will cover the basics of kinematics, dynamics and related subjects as to what sensors and actuators to include in a robot application. As a base for writing this chapter, the introductory books by Craig [2005], Niku [2001], Sciavicco and Siciliano [1996] and Selig [1992] were used, with some inspiration from Buss [2003, chapter 10]. The examples throughout this chapter were made with the help of the robotic toolbox provided by Corke [1996] for MATLAB.

The world of manipulators can be regarded as quite different from that of mobile robots: the mobile robot's odometry is generally not so good while the manipulator has a very precise knowledge of its position and orientation. On the other hand mobile robots are often equipped with several sensors perceiving information on the surrounding world, while the manipulator is often "blind" apart from the force-torque sensor. Consequently, they will face different problems while acting in a dynamic environment.

A.1 Kinematical Structure

First of all, let us see how a manipulator is mechanically designed and how this design can be expressed by parameters and equations. The kinematical structure refers to how to calculate a position of the end-point and to get the manipulator to the desired configuration. Dynamic properties, such as weight, inertia etc., are not considered as part of the kinematics.

A manipulator has one or more links. Links are connected by joints. All links are assumed to be rigid bodies, and all joints described by one degree of freedom (DOF). If a joint has more than one DOF, it is straight forward to think of it as several links with a length of zero connected with joints of one DOF. The most commonly used way to describe a manipulator is to use the following parameters:

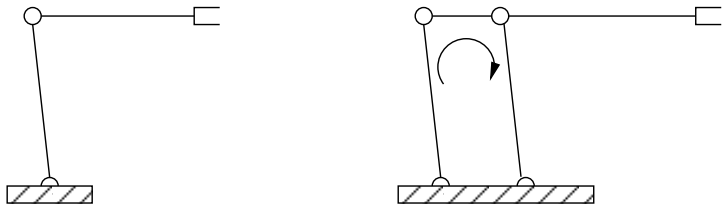


Figure A.1: *The manipulator to the left shows an open loop chain, while the manipulator to the right has closed chain structure.*

- a the length of one link,
- α the twist between two links,
- θ the rotation between two links, is variable on a revolute joint,
- d the offset distance between two joints, is variable on a prismatic joint

which are called the Denavit-Hartenberg link parameters (also known as DH representation), introduced by Hartenberg and Denavit [1955], and can be considered a de facto standard.

Almost any serial kinematical structure can be described using the Denavit-Hartenberg parameters. A very common structure of a manipulator is a *serial* design, where links are connected in a series. Other possible structures are quite common, but will only be mentioned here in brief.

A.1.1 Open and Closed Kinematic Chains

There are two design directions for manipulators; an open kinematic chain, that is when all links are in a series; or a closed chain, when both ends of two links are connected to a third link. Figure A.1 shows two examples of manipulators with the two different structures. The main reason for using the closed-chain structure is that the manipulator will be less flexible (more stiff) in the links, implying a higher degree of accuracy.

A.1.2 Rotational and Translational Joints

For the expression of rotations and translations, homogeneous transformation matrices are commonly used. Usually a joint on a robotic manipulator is of one of two different types, namely revolute or prismatic. A revolute joint is a joint which rotates about an angle θ between link n and link $n+1$, while the prismatic joint is a sliding joint that affects the link length, that is the DH-parameter a of link n.

The transformation matrix, denoted by $\mathbf{T}$, is the result of consecutive multiplications of the rotational and translational matrices:

$$\mathbf{T_n} = \mathbf{A_1 A_2 A_2 \dots A_n} \tag{A.1}$$

of n links and **A** are homogeneous rotational and translational matrices.

To express the orientation in Cartesian space, denoted by $\mathbb{R}^3$, three vectors are needed, one for each axis. For this purpose the vectors N, O and A are introduced to represent the orientational part of **T**. For the translational part, only one vector, called P, is needed, which is a conventional translation in space and only affects the position, not the orientation. The orientational part, on the other hand, is not that simple due to the fact that there is more than one way of representing an orientation (to be precise there are 24 different conventions listed by Craig [2005, appendix B]).

When combining rotation and translation they become a transformation, expressed by the homogeneous matrix:

$$\mathbf{T} = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.2}$$

The last row in the matrix is introduced due to formal reasons to facilitate matrix multiplications.

There are two different approaches to make arbitrary rotations in $\mathbb{R}^3$; around a fixed frame or around consecutive frames. Starting with the approach with consecutive frames, called *Z-Y-Z Euler angles* (illustrated to the right in figure A.2): start by applying the first rotation α to the starting frame A, then a second rotation β is applied to the resulting frame B′, yielding a new frame B″, where the last rotation γ, is applied resulting in the last frame B. Written as a series of consecutive matrix multiplications:

$${}^{A}_{B}R_{Z'Y'Z'}(\alpha,\beta,\gamma) = {}^{A}_{B'}R_Z(\alpha)\ {}^{B'}_{B''}R_Y(\beta)\ {}^{B''}_{B}R_Z(\gamma) \tag{A.3}$$

where B′ and B″ are the intermediate frames between A and B. This is equal to

$$R_{Z'Y'Z'}(\alpha,\beta,\gamma) = (R_Z(\alpha)\ R_Y(\beta))\ R_Z(\gamma) \tag{A.4}$$

where the leftmost operation is the first.

The other way is to apply all three rotations from a fixed frame (the starting frame), like the following:

$$R_{XYZ}(\gamma,\beta,\alpha) = R_Z(\alpha)\ R_Y(\beta)\ R_X(\gamma) \tag{A.5}$$

where the rotation around X comes first, followed by rotations around Y and Z consecutively (the rightmost operation is the first). Since all rotations are made from a fixed frame no intermediate frames are needed like with the Euler angles. This way of representing rotations is usually called Roll, Pitch and Yaw (RPY) angles illustrated to the left in figure A.2.

A.1.3 Forward and Inverse Kinematics

The last link of a manipulator is called the *end-effector*, and is usually a tool, for example a gripper, screwdriver, welding torch etc. This tool has a center point, called the *Tool Centre Point* or TCP, see figure A.3. From now on the name end-effector will be used, even though it could be the TCP.

Usually the end-effector point is one of the two points that are of interest in manipulating. The second is the base of the manipulator, also called the base frame. The calculation between these two points is called *kinematics* and is the tools for answering the questions "where is the end-effector with this configuration?" and "what configuration gives this end-effector position?". These two questions are the forward and inverse kinematics, respectively.

The examples in this section will be made using a 2 DOFs serial manipulator with two revolute joints. This is enough to present complete examples, avoiding rigorous and bloated formulas that will result from higher DOF manipulators.

Forward Kinematics

Normally when working with manipulators, the position and orientation of the end-effector is of more interest than the individual joints. From a given set of joint angles, it is straight forward to calculate the position and orientation of the end-effector using quite simple trigonometric functions, given a serial structure of rigid body where all link parameters are known.

A vector of n joint angles θ will be denoted by Θ. For a set of joint angles $\Theta = [\theta_1, \theta_2, \ldots \theta_n]$ the end-effector coordinates $\mathbf{x} = [x, y, z, \alpha, \beta, \gamma]$ are found with the help of the *forward kinematics*, also known as *direct kinematics*, of the robot arm. This can generally be described as:

$$\mathbf{x} = f(\Theta) \tag{A.6}$$

where $f_k(\Theta)$ is the function executing the forward kinematic mapping.

A two link planar manipulator (that means the DH-parameters α and d are zero) and rotational joints will make up a simple example. Both links have the length 1.0 units, which means the DH-parameters l_1 and l_2 are both 1.0 unit each. When θ_1 and θ_2 are zero the manipulator is aligned with the x-axis, the resulting end-effector position will then be in $l_1 + l_2 = 2.0$. Since the arm is planar, rotations can be expressed by a 3×3 matrix,

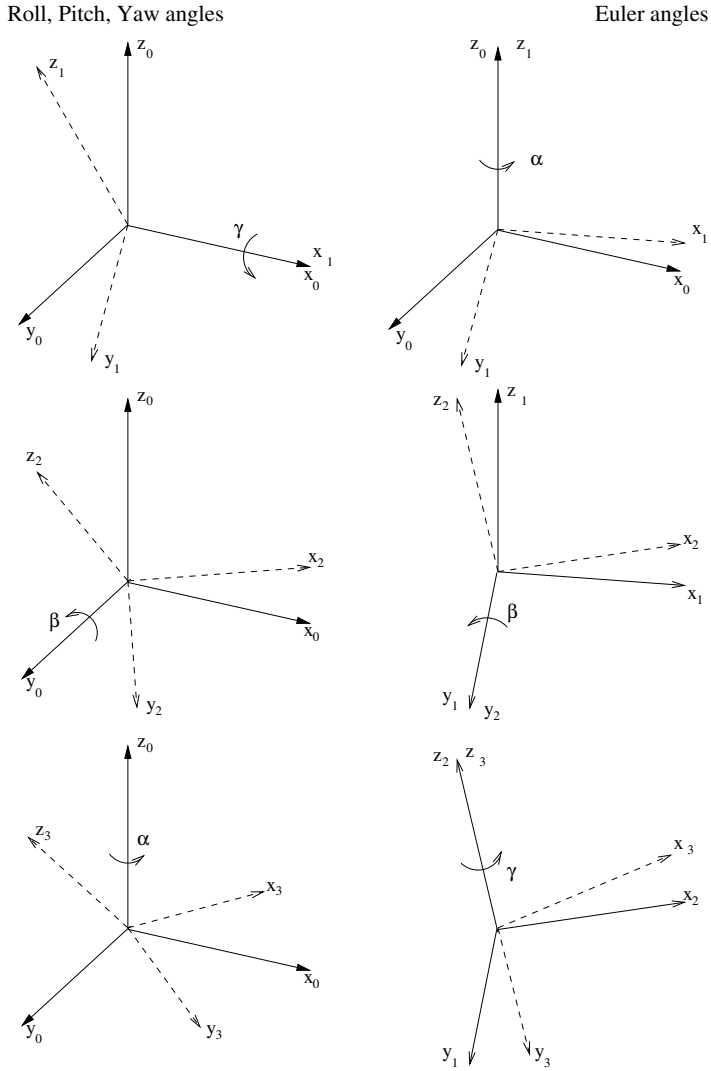


Figure A.2: *Here two rotations are shown in three steps, from top to bottom, one in each column. In the first column, the first frame, the first rotation* γ *is applied around the X-axis. Then the second rotation* β *is applied around the Y-axis, that is fixed. The final rotation* α *is then made around the fixed Z-axis. In the second column the rotations are applied around consecutive frames. In the first the rotation* α *is applied around the Z-axis* (z_1)*. In the next frame a new coordinate frame is shown with the same orientation as the last resulting rotation, where next rotation* β *is applied around* y_1 *axis in the new frame. In the last frame, the last rotation* γ *is applied around* z_2 *axis, in the latest resulting frame.*

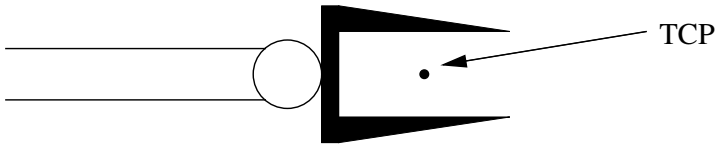


Figure A.3: *The manipulators Tool Centre Point, TCP.*

$$\mathbf{Ro}_n(\theta_n) = \begin{pmatrix} \cos\theta_n & -\sin\theta_n & 0 \\ \sin\theta_n & \cos\theta_n & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

where n is the n:th joint. Translations are also described by a 3×3 matrix,

$$\mathbf{Tr}_n(l_n) = \begin{pmatrix} 1 & 0 & l_n \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Using the arm in figure A.4 showing the manipulator in the configuration $\theta_1 = 36.87$ [deg] and $\theta_2 = -36.87$ [deg], results in the following rotational and translational matrices:

$$\mathbf{Ro}_1(36.87\text{deg}) = \begin{pmatrix} 0.80 & -0.60 & 0.00 \\ 0.60 & 0.80 & 0.00 \\ 0.00 & 0.00 & 1.00 \end{pmatrix}$$

$$\mathbf{Ro}_2(-36.87\text{deg}) = \begin{pmatrix} 0.80 & 0.60 & 0.00 \\ -0.60 & 0.80 & 0.00 \\ 0.00 & 0.00 & 1.00 \end{pmatrix}$$

$$\mathbf{Tr}_{1,2}(1.00) = \begin{pmatrix} 1.00 & 0.00 & 1.00 \\ 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 1.00 \end{pmatrix}$$

where **Ro** are the rotational matrices and **Tr** represents the translational matrix describing the link lengths. From figure A.4, first apply the rotation Ro_1, then translation Tr_1 followed by rotation Ro_a and translation Tr_2. This is:

$$\mathbf{T} = \mathbf{Ro}_1\mathbf{Tr}_1\mathbf{Ro}_2\mathbf{Tr}_2$$

where the leftmost operation is the first, resulting in:

$$\mathbf{Tr} = \begin{pmatrix} 1.00 & 0.00 & 1.80 \\ 0.00 & 1.00 & 0.60 \\ 0.00 & 0.00 & 1.00 \end{pmatrix}$$

which position and orient the TCP (see the TCP in figure A.4).

The above example used a transformation matrix $\mathbf{T}$ which is equivalent to ${}^{0}\mathbf{T}_{n}$ in equation A.1 that corresponds to the forward kinematic calculation in equation A.6. This transformation from the base to the end-effector is usually expressed as:

$${}^{0}\mathbf{T}_{n} = f_{k}(\theta_{n}) \tag{A.7}$$

where $\mathbf{T}_{n}$ is the transformation matrix, n denotes the number of DOFs and the superscript zero means the base frame.

For a general motion a manipulator requires 6 DOF, which commonly means 3 DOF to position the end-effector and 3 to orient the end-effector. Since this makes arbitrary positions and orientations possible, this is why 6 DOF is a common configuration among industrial manipulators. But it is not always necessary to have 6 DOF. For example, a pick-and-place manipulator can work with 3 DOF, or even less, depending on the task.

Inverse Kinematics

In most cases it is desired to place the end-effector at a desired position and orientation in the work space, which means that joint values resulting in such a position and orientation have to be calculated. For a 6 DOF manipulator the work space usually means a Cartesian space, which is also intuitive and convenient to use for example in planning. This leads us to the *inverse kinematics* problem. From a given point the positions of each joint is generally described by:

$$\Theta = f^{-1}(\mathbf{x}) \tag{A.8}$$

where Θ is the vector of joint angles and $\mathbf{x}$ is the work space.

For most manipulators the inverse kinematic problem can be solved analytically. However there is no general inverse kinematic solution for all manipulators that can be computed with a "one step analytical calculation", which also meets real-time requirements. For solving these calculations, there is a specific solution for each manipulator that also can take special cases into account.

The general solution to the inverse kinematic can normally be computed numerically by iteratively applying the inverse (or pseudo-inverse) of the manipulator's Jacobian:

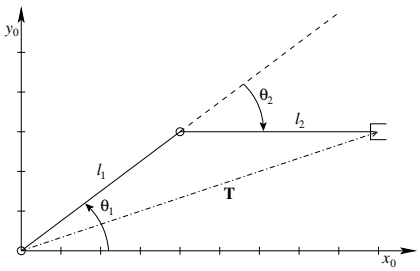


Figure A.4: *A simple planar manipulator. Simple trigonometry can be used to calculate the end-effector position.*

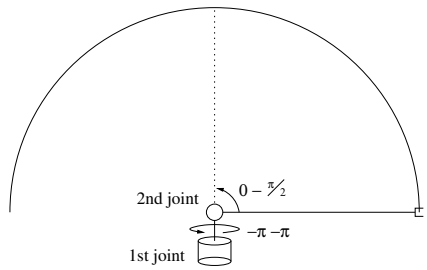


Figure A.5: *The desired arc shaped path, that the manipulator's end-effector should follow at constant velocity, after it has accelerated.*

$$\dot{\Theta} = \mathbf{J}^{-1}\dot{\mathbf{x}} \tag{A.9}$$

where $\mathbf{J}^{-1}$ denotes the inverted Jacobian. This however is not a fast technique, and can not take special cases into account. For more details on the Jacobian, see section A.3.

A.2 Singularities

If a position in work-space should be tracked, a mapping from work-space to joint-space is required, but at certain points this mapping is not possible. An example will serve as an illustration. Consider a 2 DOF manipulator with two revolute joints that follows a specified trajectory, that is a specified path with a specified velocity. This trajectory is an arc shaped path and the manipulator should track this trajectory "over its head", see to figure A.5. The x position should go from 1 to -1 and the z position from 0 to 0 via 1. It is also required to perform this tracking at a certain time, that is why it is called a trajectory. The first joint has a range from $-\pi$ to π and the range of the second joint is 0 to $\frac{\pi}{2}$. In figure A.6 the topmost graph shows the end-effector's position in x and z as a function of time. In the second graph in figure A.6, when the second joint reaches $\frac{\pi}{2}$ at time 1.0, the first joint immediately (that means at zero time) goes from 0 to π. To keep track of its trajectory, this must be done before the second joint can start going back from $\frac{\pi}{2}$ to 0. The third graph shows the velocities needed to perform this motion. It is intuitive to see that for real manipulators this is not possible due to the velocity at time 1.0, because then this point is singular.

The word *singularity* means that there is a singular point in the mapping from $\dot{\mathbf{x}} \rightarrow \dot{\Theta}$. This is the case when in the differential the inverse solution

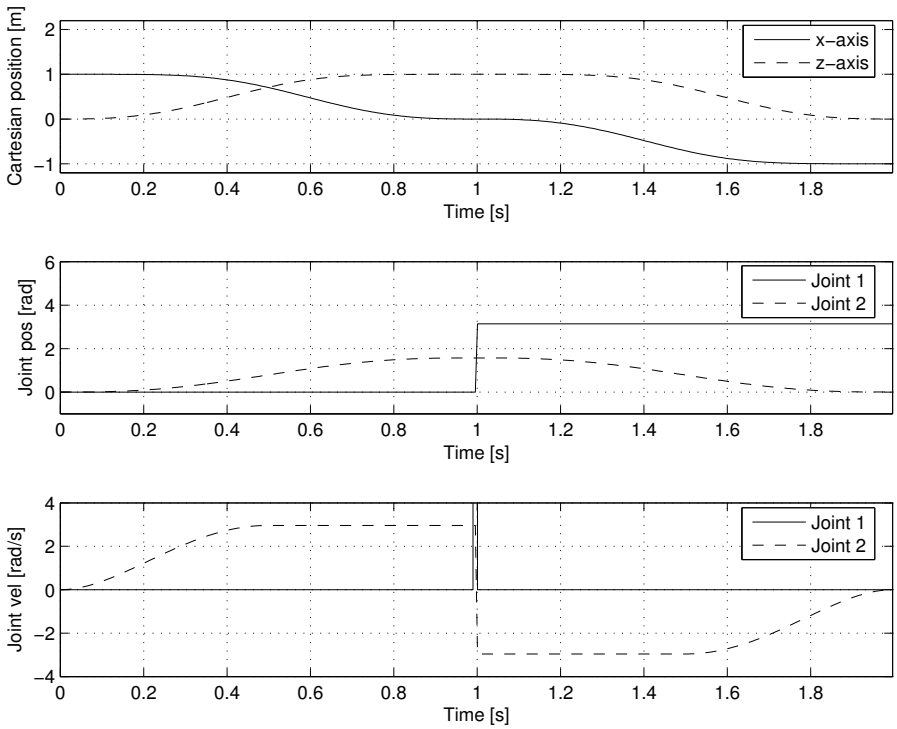


Figure A.6: *The plot at the top shows the end-effector trajectory in Cartesian space. In the middle plot the joint position is shown for both joints, and the bottom plots shows the velocities. Note the first joint's velocity at time 1.0 in the bottom plot.*

$$\dot{\Theta} = \mathbf{J}^{-1}(\Theta)\dot{\mathbf{x}} \tag{A.10}$$

does not exist. Specifically, in the singular point the determinant of $\mathbf{J}$ goes to zero

$$\det(\mathbf{J}) \to 0 \tag{A.11}$$

which results in infinite joint velocities.

The above example shows one of the problems a singularity may cause. In addition there are two other reasons to avoid singularities; there may exist an infinite number of solutions for the inverse kinematic problem and secondly, the number of possible movements from a singular point may be reduced. Think of a completely outstretched arm which only can reach just a few of the sur-

rounding points. Taking these problems into account, it is generally the case that singularities, or even to come close to them, should be avoided.

A.3 Jacobian

Mapping differential motions or velocities (motions or velocities of small magnitude) in joint space to motions or velocities in Cartesian space, is done by a transformation matrix. This matrix is called the Jacobian matrix and is similar to the forward kinematic transformation matrix, where the static vector of joint positions is transformed to a position and orientation matrix.

The differential vector represents the motion of a manipulator in joint space, denoted by $\dot{\Theta}$, and the corresponding differential motion in Cartesian space is represented by $\dot{\mathbf{x}}$. Then the Jacobian mapping in a compact matrix form is expressed by a linear mapping:

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\Theta} \tag{A.12}$$

where $\mathbf{J}$ is the Jacobian. The elements on the Jacobian are dependent of the joint variables, which means that motions in both joint space and Cartesian space are differential. Hence the Jacobian is a time dependent matrix. The mapping can be expressed more explicitly by introducing the vectors $[dx, dy, dz]^T$ and $[\delta x, \delta y, \delta z]^T$ representing the differential translational motions and rotational motions respectively. Let the joint variables $[d\theta_1, d\theta_2, \ldots, d\theta_6]^T$ represent a small motion of a 6 DOF manipulator. Equation A.12 written with these vectors then become:

$$\begin{bmatrix} dx \\ dy \\ dz \\ \delta x \\ \delta y \\ \delta z \end{bmatrix} = \begin{bmatrix} \text{Manipulator's} \\ \text{Jacobian} \end{bmatrix} \begin{bmatrix} d\theta_1 \\ d\theta_2 \\ d\theta_3 \\ d\theta_4 \\ d\theta_5 \\ d\theta_6 \end{bmatrix} \tag{A.13}$$

Going back to a more compact form that is valid for the general case, that means any number of DOF of the manipulator, and also include the time dependency, each vector and the Jacobian matrix can be expressed as:

$$\delta\mathbf{x} = [dx,\ dy,\ dz,\ \delta x,\ \delta y,\ \delta z]^T \tag{A.14}$$

$$J(\Theta) = \left[\begin{array}{c} \text{Manipulator's} \\ \text{Jacobian} \end{array} \right] \tag{A.15}$$

$$\delta\Theta = [d\theta_1,\ d\theta_2,\ \dots\ d\theta_n\ ]^T \tag{A.16}$$

Here Θ represents the joint positions, $\delta\Theta$ and $\delta\mathbf{x}$ the differential motions in joint space and Cartesian space respectively, $J(\Theta)$ represents the Jacobian dependent of Θ and n is the number of DOF. Using the general notation from equation A.14 to A.16 in the form of equation A.12 yields:

$$\delta\mathbf{x} = J(\Theta)\delta\Theta \tag{A.17}$$

Since the Jacobian is a matrix of partial derivatives of functions of Θ, equation A.17 can be rewritten to:

$$\delta\mathbf{x} = \frac{\partial f}{\partial\Theta}\delta\Theta \tag{A.18}$$

where f would be:

$$\mathbf{x} = f_k(\Theta)$$

which is the forward kinematic equation A.6. Each function $\frac{\partial f}{\partial\Theta}$ that is being derived has a number of variables, one for each degree of freedom (in previous equation A.13 six were used). To be more general the number of joints is denoted n, the notation $\mathbf{q}$ is used, instead of Θ, representing both revolute and prismatic joints. The Cartesian space is represented by m components, where $1 \leqslant m \leqslant 6$. The forward kinematic equation is then calculated with respect to each joint:

$$\mathbf{x}_j = f_j(\mathbf{q}) \tag{A.19}$$

where $\mathbf{x}_j = [x, y, z, \alpha, \beta, \gamma]$, $\mathbf{q} = [q_1, q_2, \cdots, q_n]$ and $f_j = [f_1, f_2, \cdots, f_m]$. Calculating the differential change of $\mathbf{x}$ of the differential change in $\mathbf{q}$ using equation A.19 gives:

$$\begin{aligned}
\delta \mathbf{x}_1 &= \frac{\partial f_1}{\partial q_1}\delta q_1 + \frac{\partial f_1}{\partial q_2}\delta q_2 + \cdots + \frac{\partial f_1}{\partial q_n}\delta q_n, \\
\delta \mathbf{x}_2 &= \frac{\partial f_2}{\partial q_1}\delta q_1 + \frac{\partial f_2}{\partial q_2}\delta q_2 + \cdots + \frac{\partial f_2}{\partial q_n}\delta q_n, \\
&\vdots \\
\delta \mathbf{x}_m &= \frac{\partial f_m}{\partial q_1}\delta q_1 + \frac{\partial f_m}{\partial q_2}\delta q_2 + \cdots + \frac{\partial f_m}{\partial q_n}\delta q_n
\end{aligned} \tag{A.20}$$

Putting equation A.20 in matrix form results in the following:

$$\begin{bmatrix} \delta \mathbf{x}_1 \\ \delta \mathbf{x}_2 \\ \vdots \\ \delta \mathbf{x}_m \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \frac{\partial f_1}{\partial q_2} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \frac{\partial f_2}{\partial q_1} & \frac{\partial f_2}{\partial q_2} & \cdots & \frac{\partial f_2}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial q_1} & \frac{\partial f_m}{\partial q_2} & \cdots & \frac{\partial f_m}{\partial q_n} \end{bmatrix} \begin{bmatrix} \delta q_1 \\ \delta q_2 \\ \vdots \\ \delta q_n \end{bmatrix} \tag{A.21}$$

which is an explicit expression for each element of the Jacobian.

A.3.1 Inverse Jacobian

For mapping small differential velocities and motions in Cartesian space into velocities or motions in joint space it is necessary to compute the inverted Jacobian. Starting from equation A.12, the inverted Jacobian $\mathbf{J}^{-1}$ is derived from:

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\Theta}$$

$$\mathbf{J}^{-1}\dot{\mathbf{x}} = \mathbf{J}^{-1}\mathbf{J}\dot{\Theta} \rightarrow$$

$$\dot{\Theta} = \mathbf{J}^{-1}\dot{\mathbf{x}} \tag{A.22}$$

One problem with this mapping arises from the fact that the Jacobian can have an arbitrary number of columns, each corresponding to one joint. The result might be a non-squared matrix, that is not directly invertible. In such cases the pseudo inverse Jacobian, denoted $\mathbf{J}^{+}$, can be used instead. However, if the inverse does not exist, this technique does not work, and in general it is computationally heavy to perform matrix inversion.

Depending on the Jacobian's configuration, there are two ways to compute the pseudo inverse. Let the Jacobian be an $m \times n$ matrix, that is a manipulator with n joints, where $m \neq n$, then the pseudo inverted Jacobian is:

$$\mathbf{J}^{+} = \mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1}, \quad \text{if } m < n \tag{A.23}$$

$$\mathbf{J}^{+} = (\mathbf{J}^{T}\mathbf{J})^{-1}\mathbf{J}^{T}, \quad \text{if } m > n \tag{A.24}$$

where equation A.23 is valid when the manipulator has more DOF than is required for the task. The other case, when the manipulator has less DOF than is required for the task, equation A.24 is valid.

A.4 Trajectory Planning and Generation

A path can be distinguished from a trajectory by determine whether the time parameter is of importance or not. If a certain point on a path has to be met at a specific time the desired motion is called a *trajectory*, otherwise it is called a path.

To perform a motion an initial point t_s and a goal point t_f are specified, which can be done in joint space or in Cartesian space. When moving the end-effector from the initial point to the goal point, the manipulator's controller needs to plan this motion. The most intuitive way to go from point A to point B is to follow a straight line in Cartesian space. However a straight line in Cartesian space rarely means a straight line in joint space. In addition a motion is also often desired to be smooth, which means that the first two derivatives exist.

A simple example is a two link planar manipulator with both links of length 1 (the same as previously shown in figure A.4). A straight line path from the initial point $t_s = [1.3\ -1.0]$ to the goal point $t_f = [1.7\ 1.0]$ is made in Cartesian space, as in figure A.7 A. The resulting motion in joint space is a non-linear curve, seen in figure A.7 B. If the same motion was planned as a linear motion in joint space, figure A.7 D, the path of the end-effector will follow a non-linear path in Cartesian space, shown in figure A.7 C.

In order to draw a straight line path in Cartesian space, from an arbitrary point A to an arbitrary point B, the path must be divided into several small segments. When the segments are made small enough the previously described Jacobian can be applied to make these small changes, since the Jacobian is a differential mapping applicable to differences both in positions and velocities.

When performing trajectory planning in Cartesian space a number of problems are introduced, namely:

Intermediate points Some of the intermediate points on the path are unreachable, see figure A.8 A.

Different solutions In cases when all points on a path are reachable but from different configurations, which means that the path as a whole can not be followed, see figure A.8 B.

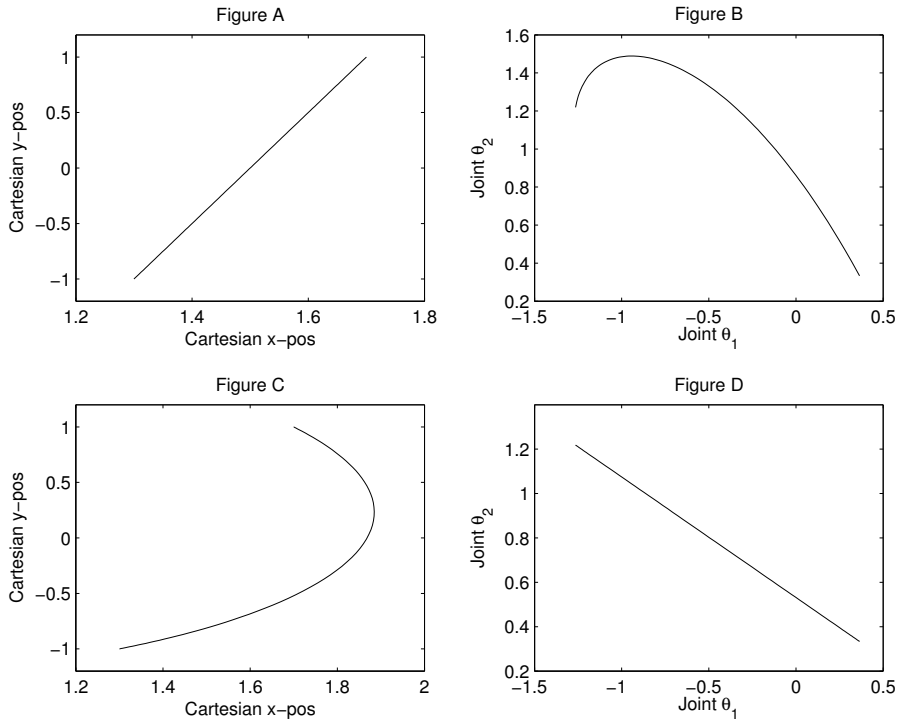


Figure A.7: *The trajectory shown in A is a straight line in Cartesian space, but non-linear in joint space (figure B). In figure D the trajectory planning is made linear in joint space, and results in a curved trajectory in Cartesian space.*

Singularities Close to singularities the joint velocity become undesirably high, see figure A.8 C. In section A.2 this were discussed, also compare to figure A.6.

Applications that require a specified trajectory in Cartesian space, for example arc welding, can not be planned in joint space. Therefore restrictions have to be taken into account when planning a trajectory in Cartesian space. In many other applications it can be suitable to plan the path in joint space.

A.4.1 Polynomial Trajectory Planning

To follow a path in Cartesian space, for example a straight line motion, many points must be specified at the desired resolution. However, in most cases a path or trajectory is specified with more than just an initial point and a goal point, but rather than specifying "all" points (as for a straight line) a few important intermediate points are selected. These points are called *via points*, and provide

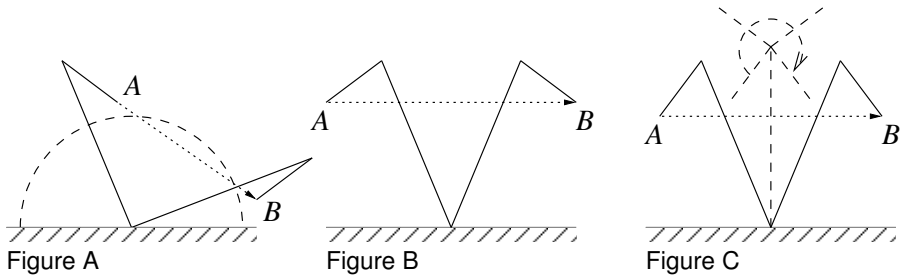


Figure A.8: *Figure A shows a path where the points inside the dashed circle are unreachable. In figure B all points on the path are reachable, but since a different configuration is required in the start position A than in the goal position B the planner can not follow the specified path. Figure C shows a similar problem, where points close to a singularity will cause high joint rates, indicated by the dashed figure.*

a parametric description of a trajectory. To make use of these via points polynomials can be used, also known as *cubic polynomial*, which generate smooth trajectories. Polynomials are only applicable to trajectories in joint space and not in Cartesian space, since they specify the joint coordinates and not the Cartesian coordinates. A third order polynomial is:

$$\theta(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 \tag{A.25}$$

with the first and second derivatives:

$$\dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2 \tag{A.26}$$

$$\ddot{\theta}(t) = 2c_2 + 6c_3 t \tag{A.27}$$

Usually in most motions the following is known: 1) the starting position, 2) the end position, 3) the initial velocity and, 4) the velocity at the end of the movement. This can be described by:

$$\begin{array}{rcl} \theta(t_s) & = & \theta_s \\ \theta(t_f) & = & \theta_f \\ \dot{\theta}(t_s) & = & 0 \\ \dot{\theta}(t_f) & = & 0 \end{array} \tag{A.28}$$

where t_s is the starting time, t_f is the end time, θ_s the starting position, θ_f the goal position and assuming the start and goal velocities to be zero. Putting these

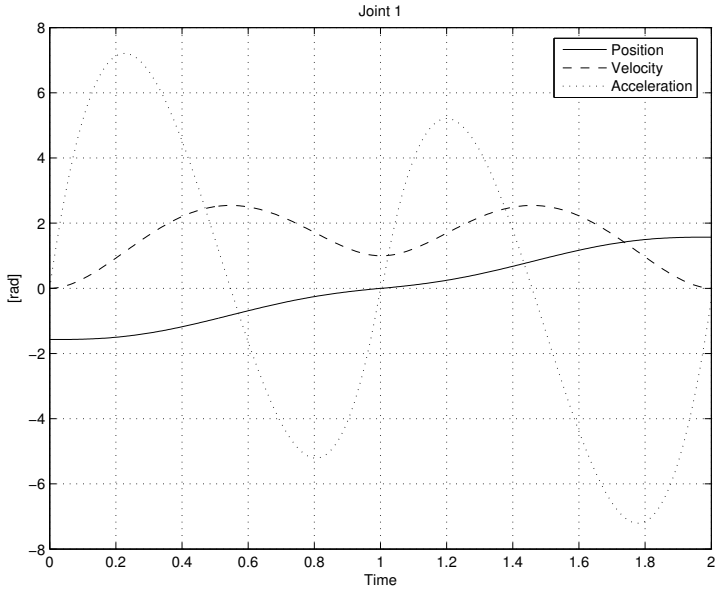


Figure A.9: *At time 1.0 the acceleration is* 0*. This trajectory is made with a 7:th order polynomial.*

four assumptions into the polynomial in A.25 and A.26 yields the following equations:

$$\begin{array}{rcl} \theta(t_s) & = & c_0 = \theta_s \\ \theta(t_f) & = & c_0 + c_1 t_f + c_2 t_f^2 + c_3 t_f^3 = \theta_f \\ \dot{\theta}(t_s) & = & c_1 = 0 \\ \dot{\theta}(t_f) & = & c_1 + 2c_2 t_f + 3c_3 t_f^2 = 0 \end{array} \tag{A.29}$$

Knowing the parameters $\theta(t_s), \theta(t_f), \dot{\theta}(t_s)$ and $\dot{\theta}(t_f)$ in advance one can solve the four unknown elements $c_0 \ldots c_3$.

The assumptions that the initial and final velocities are zero are true for many motions, but is not the general case. If a start and/or goal position is desired to be a via point where the motion is indented to continue, it is possible to set the velocity arbitrarily. In order to specify the acceleration the third order polynomial is extended to a fifth order polynomial, with the corresponding first and second derivatives:

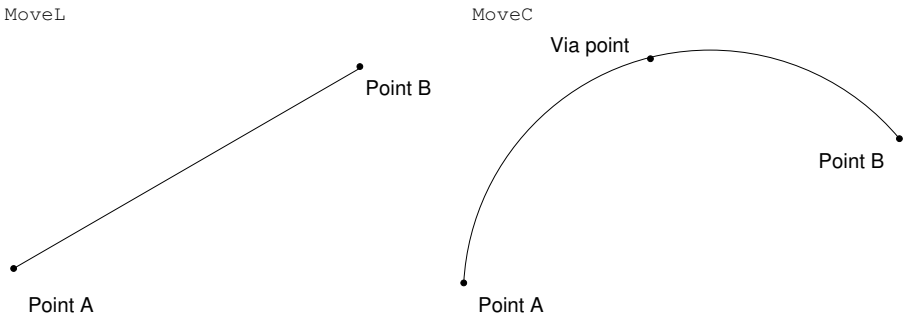


Figure A.10: *Example paths produced by two different commands using ABB's programming language RAPID. The left is a straight line path* (MoveL)*, and the right is a circular path* (MoveC)*. Both paths are shown in Cartesian space.*

$$\theta(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5 \tag{A.30}$$

$$\dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2 + 4c_4 t^3 + 5c_5 t^4 \tag{A.31}$$

$$\ddot{\theta}(t) = 2c_2 + 6c_3 t + 12c_4 t^2 + 20c_5 t^3 \tag{A.32}$$

Having second order derivatives that are continuous will generate smooth trajectories, which will relieve the mechanics from unnecessary stress. By using polynomials of high order it is also possible to incorporate constraints into the equations. However, higher order polynomials mean higher computational requirements which is an unwanted property when doing real-time calculations. Instead of using higher order polynomials, it is better to combine several polynomials of lower order. Many other techniques can be applied instead of polynomials, such as splines, but no further details will be presented here.

As an example of trajectory generation a joint space trajectory is specified (shown in figure A.9), from $-\frac{\pi}{2}$ via 0.0 to $\frac{\pi}{2}$, with the velocity set to 1.0 at the via point and the via point should be reached at time 1.0 s.

A.4.2 Trajectory Planning in Cartesian Space

For industrial manipulators trajectory planning in Cartesian space is usually a part of the programming language. The programmer specifies the points using commands like:

`MoveL` The robot moves the TCP from the current position, to the destination in a straight (linear) line in Cartesian space. This motion is shown to the left in figure A.10.

`MoveC` Moves the TCP from the current position, through a desired point, to a destination in a circular movement in Cartesian space. The arc shape of

the circular movement is determined by the desired point and the destination. This motion is shown to the right in figure A.10.

`MoveJ` The robot moves the end-effector to specified point, all joint movement reaches the destination at the same time. This movement is not a straight line in Cartesian space. Compare this to figure A.7, where joints' motions are linear but the TCP moves in an arc.

The above commands are from the RAPID programming language (see the manual from ABB Robotics) which all moves the tool mounted on the manipulator in different ways, but all motions are specified in Cartesian space. The programmer does not have to bother about converting the motion into joint space, the system takes care of this.

A.5 Robot Control

From the trajectory planning and generation in the previous section, the desired values of each joint were obtained, in this section referred to as θ_d. Since the manipulator like any other machine is affected by internal and external disturbances and dynamics, the desired joint value and the actual joint value will differ and produce an error, $e = \theta - \theta_d$. This means a control system is needed. Like dynamics, control is in itself a scientific field, and below, some basic control mechanisms used in robotics are presented.

Control of the robotic manipulator can either be applied to joint space or in Cartesian space. Joint space control is the one used in industrial robots today, mostly using an independent-joint PID controller [Craig, 2005, chapter 9]. But there are several different ways of controlling a manipulator; an overview is presented by An et al. [1988, chapter 1]. The three most commonly used are:

- independent-joint proportional-differential (PID) control,
- feedforward-control, and
- computed torque control,

of which the first, independent-joint PID control, is the most popular. The proportional gain, P in PID, is an error minimization gain term that removes a portion of the error. The derivative component D has a damping effect, and inhibits overshooting, that is, if the error is minimized too much (increasing in the other direction). Finally, I refers to the integral term, which cancels the steady state error, that is when the system reaches an equilibrium point [Åström and Hägglund, 1995, chapter 3]. Feedforward means that a prediction of the future control value is made (possibly by an internal model).

A.5.1 Position Control

Positional control, that is when the controller tries to keep the manipulator in the desired position. In the independent-joint PD control each joint is controlled individually, hence the name, and the control actions are completely independent of each other. However the name can be misleading because the joint motions are highly coupled. The torque, τ, is given by:

$$\tau = K_p(\Theta - \Theta_d) + K_v\dot{\Theta} \tag{A.33}$$

where the error, $e = \Theta - \Theta_d$, between the desired value, Θ_d and the true value Θ, together with the velocity $\dot{\Theta}$ sets the torque. The two gain diagonals matrices K_p and K_v are constant, and represents the positional and velocity gain respectively.

A.5.2 Trajectory Following

If the desired velocity is introduced, that is $\dot{\Theta}_d$, the system becomes time dependent. The basic PD control law can easily be extended to velocity error, that is $\dot{e} = \dot{\Theta} - \dot{\Theta}_d$, forming the equation:

$$\tau = K_p(\Theta - \Theta_d) + K_v(\dot{\Theta} - \dot{\Theta}_d) \tag{A.34}$$

Furthermore, this can be extended to also have an integral part, finally arriving at the earlier mentioned independent-joint PID controller:

$$\tau = K_p(\Theta - \Theta_d) + K_i \int^t (\Theta - \Theta_d)\, dt + K_v(\dot{\Theta} - \dot{\Theta}_d) \tag{A.35}$$

where the middle term is the integrated error over time.

If there is a model of the manipulator, it can be beneficial to incorporate this into the controller. Using the PD-control law (equation A.34) and extending it with the model of the manipulator makes it a model-based feedforward controller, described by An et al. [1988] and Craig [2005], yield the equation:

$$\tau = \hat{R}^{-1}(\Theta_d, \dot{\Theta}_d, \ddot{\Theta}_d) + K_p(\Theta - \Theta_d) + K_v(\dot{\Theta} - \dot{\Theta}_d) \tag{A.36}$$

where $\hat{R}^{-1}(\Theta_d, \dot{\Theta}_d, \ddot{\Theta}_d)$ is the dynamic model of the robot. The torque is computed as a function of the desired path, and can be computed off-line in advance if the path is known.

References

ABB Robotics. *RAPID Reference Manual*. ABB Robotics AB, S-721 68 Västerås, Article number: 3HAC7774-1.

A. A. Ahmed, D. M. Wolpert, and J. R. Flanagan. Flexible representations of dynamics are used in object manipulation. *Current Biology*, 18(10):763–768, 2008. doi: 10.1016/j.cub.2008.04.061.

J. Aleotti. *Programming by Demonstration in Virtual Reality*. PhD thesis, University of Parma, March 2006.

J. Aleotti and S. Caselli. Robust trajectory learning and approximation for robot programming. *Robotics and Autonomous Systems*, 54(5):409–413, 2006. doi: 10.1016/j.robot.2006.01.003.

J. Aleotti, S. Caselli, and M. Reggiani. Leveraging on a virtual environment for robot programming by demonstration. *Robotics and Autonomous Systems*, 47(2–3):153–161, 2004. doi: 10.1016/j.robot.2004.03.009.

C. H. An, C. G. Atkeson, and J. M. Hollerbach. *Model-based control of a robot manipulator*, chapter 1, pages 16–20. Cambridge, MA: MIT Press., 1988.

A. Ananiev, I. Kalaykov, and B. Iliev. An approach to the design of lightweight reconfigurable robot arm for a mobile platform. In *International Symposium of Robotics*, October 2002.

M. Asada, K. F. MacDorman, H. Ishiguro, and Y. Kuniyoshi. Cognitive developmental robotics as a new paradigm for the design of humanoid robots. *Robotics and Autonomous Systems*, 37(2-3):185—193, 2001. doi: 10.1016/S0921-8890(01)00157-9.

C. G. Atkeson and S. Schaal. Robot learning from demonstration. In *ICML*, pages 12–20, 1997.

C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997a.

C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. *Artificial Intelligence Review*, 11(1-5):75–113, 1997b.

D. C. Bentivegna, C. G. Atkeson, and G. Cheng. Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47(2–3):163–169, 2004.

G. Biggs and B. MacDonald. A survey of robot programming systems. In *Proceedings of the Australian Conference on Robotics and Automation*, 2003.

A. Billard, Y. Epars, S. Calinon, S. Schaal, and G. Cheng. Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2–3):69–77, 2004. doi: 10.1016/j.robot.2004.03.002.

E. Bizzi, F. A. Mussa-Ivaldi, and S. Giszter. Computations underlying the execution of movement: A biological perspective. *Science*, 253(5017):287–291, July 1991.

E. Bizzi, S. F. Giszter, E. Loeb, F. A. Mussa-Ivaldi, and P. Saltiel. Modular organization of motor behavior in the frog's spinal cord. *Trends in Neuroscience*, 18(10):442–446, 1995.

M. Brass and C. Heyes. Imitation: is cognitive neuroscience solving the correspondence problem? *Trends in Cognitive Sciences*, 9(10):489–495, October 2005. doi: 10.1016/j.tics.2005.08.007.

D. Bullock and S. Grossberg. VITE and FLETE: Neural modules for trajectory formation and postural control. In W. A. Hershberger, editor, *Volitonal Action*, pages 253–297. Elsevier Science Publishing B.V. (North-Holland), 1989.

S. R. Buss. *3D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, 2003.

R. W. Byrne. Imitation as behaviour parsing. *Philosophical Transaction of the Royal Society of London: Series B, Biological Sciences*, 358(1431):529–536, March 2003. doi: 10.1098/rstb.2002.1219.

S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(2):286–298, April 2007. doi: 10.1109/TSMCB.2006.886952.

K. Charusta, D. Dimitrov, A. J. Lilienthal, and B. Iliev. Grasp-related features extraction by human dual-hand object exploration. In *Submitted to the 14:th International Conference on Advanced Robotics*, 2009.

M. Conditt and F. A. Mussa-Ivaldi. Central representation of time during motor learning. In *Proc. Natl. Acad. Sci. USA*, volume 96, pages 11625–11630, September 1999.

P. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, Mar. 1996.

J. J. Craig. *Introduction to Robotics - Mechanics and Control*. Prentice Hall International, third edition, 2005.

R. H. Cuijpers, H. T. van Schie, M. Koppen, W. Erlhagen, and H. Bekkering. Goals and means in action observation: A computational approach. *Neural Networks*, 19:311–322, 2006. doi: 10.1016/j.neunet.2006.02.004.

K. Dautenhahn and C. Nehaniv, editors. *Imitation in animals and artifacts*. The MIT Press, Cambridge, MA, 2002a.

K. Dautenhahn and C. L. Nehaniv. The agent-based perspective on imitation. In K. Dautenhahn and C. L. Nehaniv, editors, *Imitation in Animals and Artifacts*. MIT Press, 2002b.

R. Dawkins. *The God Delusion*. Transworld Publishers, 2006.

N. Delson and H. West. Robot programming by human demonstration: The use of human variation in identifying obstacle free trajectories. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 564–571, Sept 1994. doi: 10.1109/IROS.1994.407519.

N. Delson and H. West. Robot programming by human demonstration: Adaptation and inconsistency in constrained motion. In *IEEE International Conference on Robotics and Automation*, pages 30–36, 1996. doi: 10.1109/ROBOT.1996.503569.

R. Dillmann. Teaching and learning of robot tasks via observation of human performance. *Robotics and Autonomous Systems*, 47(2–3):109–116, 2004.

K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12:219–245, 2000.

R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, New York, 2nd edition, 2000.

B. Dufay and J. Latombe. An approach to automatic robot programming based on inductive learning. *The International Journal of Robotic Research*, 3(4): 3–20, 1984.

M. Ehrenmann, R. Zöllner, O. Rogalla, and R. Dillmann. Programming service tasks in household environments by human demonstration. In *Proceedings of the 11th IEEE International Workshop on Robot and Human Interactive Communication (ROMAN)*, Berlin, Sep. 22-27 2002.

S. Ekvall. *Robot Task Learning from Human Demonstration*. PhD thesis, KTH School of Computer Science and Communication, 2007.

S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *IEEE International Symposium on Robot and Human Interactive Communication*, Seattle, USA, 2006. IEEE.

W. Erlhagen and E. Bicho. The dynamic neural field approach to cognitive robotics. *Journal of Neural Engineering*, 3:R36–R54, 2006. doi: 10.1088/1741-2560/3/3/R02.

W. Erlhagen, A. Mukovskiy, E. Bicho, G. Panin, C. Kiss, A. Knoll, H. von Schie, and H. Bekkering. Goal-directed imitation for robots: A bio-inspired approach to action understanding and skill learning. *Robotics and Autonomous Systems*, 54(5):353–360, 2006. doi: 10.1016/j.robot.2006.01.004.

S. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufman, 1990.

P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6): 381–391, June 1954.

T. Flash and N. Hogan. The coordination of arm movements: An experimentally confirmed mathematical model. *The Journal of Neuroscience*, 5(7): 1688–1703, 1985.

A. Fod, M. J. Matatić, and O. C. Jenkins. Automated derivation of primitives for movment classification. *Autonomous Robots*, 12(1):39–54, 2002.

H. Friedrich, S. Münch, R. Dillmann, S. Bocionek, and M. Sassin. Robot programming by demonstration: Supporting the induction by human interaction. *Machine Learning*, pages 163–189, May 1996.

V. Gullapalli. Learning under extreme uncertainty. *Advances in Neural Information Processing*, 5:327–334, 1993.

D. Gustafson and W. Kessel. Fuzzy clustering with a fuzzy covariance matrix. *Proceedings of the 1979 IEEE CDC*, pages 761–766, 1979.

S. H. G. ten Hagen. *Continuous State Space Q-Learning for Control of Nonlinear Systems*. PhD thesis, University of Amsterdam, Feb 2001.

C. M. Harris and D. M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394(6695):780–785, Aug 1998.

R. S. Hartenberg and J. Denavit. A kinematic notation for lower pair mechanisms based on matrices. *Journal of Applied Mechanics*, 77:215–221, June 1955.

M. Hersch and A. G. Billard. A biologically-inspired controller for reaching movements. In *Proc. IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics*, pages 1067–1071, Pisa, 2006. doi: 10.1109/BIOROB.2006.1639233.

M. Hersch and A. G. Billard. Reaching with multi-referential dynamical systems. *Autonomous Robots*, 25(1–2):71–83, August 2008. doi: 10.1007/s10514-007-9070-7.

M. Hersch, F. Guenter, S. Calinon, and A. G. Billard. Learning dynamical system modulation for constrained reaching tasks. In *IEEE-RAS International Conference on Humanoid Robots*, pages 444–449, 2006. doi: 10.1109/ICHR.2006.321310.

R. Huys, B. E. Studenka, N. L. Rheaume, H. N. Zelaznik, and V. K. Jirsa. Distinct timing mechanisms produce discrete and continuous movements. *PLoS Comput Biol*, 4(4):e1000061, apr 2008. doi: 10.1371/journal.pcbi.1000061. URL `http://dx.doi.org/10.1371%2Fjournal.pcbi.1000061`.

T. Iberall. Human prehension and dexterous robot hands. *The International Journal of Robotics Research*, 16(3):285–299, 1997. doi: 10.1177/027836499701600302.

A. Ijspeert, J. Nakanishi, and S. Schaal. Trajectory formation for imitation with nonlinear dynamical systems. In *IEEE International Conference on Intelligent Robots and Systems (IROS 2001)*, volume 2, pages 752–757, 2001. doi: 0.1109/IROS.2001.976259.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pages 1398–1403, 2002. doi: 10.1109/ROBOT.2002.1014739.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In S. Becker, S. Thrun, and O. K., editors, *Advances in Neural Information Processing Systems 15*, 2003.

B. Iliev, B. Kadmiry, and R. Palm. Interpretation of human demonstrations using mirror neuron system principles. In *Proceedings of the 6th IEEE International Conference on Development and Learning*, pages 128–133, Imperial College London, 11-13 July 2007. doi: 10.1109/DEVLRN.2007.4354036.

I. Iossifidis and G. Schöner. Autonomous reaching and obstacle avoidance with the anthropomorphic arm of a robotic assistant using the attractor dynamics approach. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, April 2004.

I. Iossifidis and G. Schöner. Dynamical systems approach for the autonomous avoidance of obstacles and joint-limits for an redundant robot arm. In *Proceedings of 2006 IEEE International Conference on Robotics and Automation*, pages 580–585, 2006. doi: 10.1109/IROS.2006.282468.

M. Ito, K. Noda, Y. Hoshino, and J. Tani. Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model. *Neural Networks*, 19(3):323–337, 2006. doi: 10.1016/j.neunet.2006.02.007.

B. Jansen and T. Belpaeme. A computational model of intention reading in imitation. *Robotics and Autonomous Systems*, 54:394–402, 2006. doi: 10.1016/j.robot.2006.01.006.

Z. Ju, H. Liu, X. Zhu, and Y. Xiong. Dynamic grasp recognition using time clustering, gaussian mixture models and hidden markov models. In C. Xiong, editor, *Intelligent Robotics and Applications*, pages 669–678. Springer, 2008.

M. Kaiser, H. Friedrich, and R. Dillmann. Obtaining Good Performance from a bad Teacher. In *Workshop: Programming by Demonstration vs. Learning from Examples; International Conference on Machine Learning*, 1995.

Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10(6):799–822, 1994.

Y. Kuniyoshi, Y. Yorozu, M. Inaba, and H. Inoue. From visuo-motor self learning to early imitation - a neural architecture for humanoid learning. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, volume 3, pages 3132–3139, 2003. doi: 10.1109/ROBOT.2003.1242072.

F. Lacquaniti, C. Terzuolo, and P. Viviani. The law relating the kinematic and figural aspects of drawing movements. *Acta Psychologica*, 54(1-3):115–130, 1983.

W. S. Levine. The root locus plot. In W. S. Levine, editor, *The Control Handbook*, chapter 10.4, pages 192–198. CRC Press, 1996.

J. Li and T. Duckett. Growing RBF networks for learning reactive behaviours in mobile robotics. *International Journal of Vehicle Autonomous Systems (IJVAS) Special Issue on: "Computational Intelligence and Its Applications to Mobile Robots and Autonomous System"*, 2005.

J. Liebermann and C. Breazeal. Improvements on action parsing and action interpolation for learning though demonstration. In *IEEE International Conference on Humanoid Robots*, volume 1, pages 342–365, Los Angeles, CA, 10-12 Nov. 2004.

A. Liegeois. Automatic supervisory control of the configuration and behaviour of multibody mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(12):868–871, 1977.

M. Lopes and J. Santos-Victor. A developmental roadmap for learning by imitation in robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(2):308–321, April 2007. doi: 10.1109/TSMCB.2006.886949.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

F. A. Mussa-Ivaldi and E. Bizzi. Motor learning through the combination of primitives. *Philosophical Transaction of the Royal Society of London: Series B, Biological Sciences*, 355:1755–1769, 2000. doi: 10.1098/rstb.2000.0733.

F. A. Mussa-Ivaldi, S. F. Giszter, and E. Bizzi. Linear combinations of primitives in vertebrate motor control. *Proc. Natl. Acad. Sci.*, 91(16):7534–7538, 1994.

C. L. Nehaniv and K. Dautenhahn. The correspondence problem. In Dautenhahn and Nehaniv [2002a], pages 41–61.

S. B. Niku. *Intruduction to Robotics - Analysis, Systems, Applications*. Prentice Hall, Inc., 2001.

K. Ogawara, J. Takamatsu, H. Kimura, and K. Ikeuchi. Extraction of essential interactions through multiple observations of human demonstrations. *IEEE Transactions on Industrial Electronics*, 50(4):667–675, 2003. doi: 10.1109/TIE.2003.814765.

E. Oztop and M. A. Arbib. Schema design and implementation of the grasp-related mirror neurons. *Biological Cybernetics*, 87(2):116–140, 2002. doi: 10.1007/s00422-002-0318-1.

R. Palm. Control of a Redundant Manipulator Using Fuzzy Rules. *Fuzzy Sets and Systems*, 45(3):279–298, feb 1992.

R. Palm and B. Iliev. Learning of grasp behaviors for an artificial hand by time clustering and Takagi-Sugeno modeling. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 291–298, Vancouver, BC, Canada, July 16-21 2006. doi: 10.1109/fuzzy.2006.1681728.

R. Palm and B. Iliev. Segmentation and recognition of human grasps for programming-by-demonstration using time-clustering and fuzzy modeling. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, London, UK, July 23-26 2007.

R. Palm and C. Stutz. Generation of control sequences for a fuzzy gain scheduler. *International Journal of Fuzzy Systems*, 5(1):1–10, March 2003.

R. Palm, D. Driankov, and H. Hellendoorn. *Model Based Fuzzy Control*. Springer, 1997.

M. Pardowitz, S. Knoop, R. Dillmann, and R. D. Zöllner. Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 37(2):322–332, April 2007. doi: 10.1109/TSMCB.2006.886951.

J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Third IEEE-RAS International Conference on Humanoid Robots*, 2003.

C. Rigotti, P. Cerveri, G. Andreoni, A. Pedotti, and G. Ferrigno. Modeling and driving a reduced human mannequin through motion captured data: A neural network approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 31(3):187–193, May 2001. doi: 10.1109/3468.925658.

G. Rizzolatti. The mirror neuron system and its function in humans. *Anatomy and Embryology*, 210(5):219–221, October 2005. doi: 10.1007/s00429-005-0039-z.

G. Rizzolatti, L. Fadiga, V. Gallese, and L. Fogassi. Premotor cortex and the recognition of motor actions. *Cognitive Brain Research*, 3:131–141, 1996.

D. A. Rosenbaum, C. M. van Heugten, and G. E. Caldwell. From cognition to biomechanics and back: The end-state comfort effect and the middle-is-faster effect. *Acta Psychologica*, 94(1):59–85, 1996.

K. Samejima, K. Katagiri, K. Doya, and M. Kawato. Symbolization and imitation learning of motion sequence using competitive modules. *Electronics and Communications in Japan, Part 3*, 89(9):42–53, 2006. doi: 10.1002/ecjc.20267.

S. Schaal. Learning from demonstration. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 1040–1046. The MIT Press, 1997.

S. Schaal. Is imitation learning the route to humanoids? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

S. Schaal, C. G. Atkeson, and S. Vijayakumar. Real-time robot learning with locally weighted statistical learning. In R. Jarvis and A. Zelinsky, editors, *International Symposium on Robotic Research*, 2001.

S. Schaal, C. G. Atkeson, and S. Vijayakumar. Schalable techniques from nonparametric statistics for real-time robot learning. *Applied Intellicence*, 17(1): 49–60, 2002.

L. Sciavicco and B. Siciliano. *Modeling and Control of Robot Manipulators*. The McGraw-Hill Companies, Inc., 1996.

J. M. Selig. *Introductory Robotics*. Prentice Hall International (UK), 1992.

R. Shadmehr and S. P. Wise. *Computational Neurobiology of Reaching and Pointing - A Foundation for Motor Learning*. Computational Neuroscience. The MIT Press, 2005.

A. Sharkey. On combining artificial neural nets. In *Connection Science*, volume 8, pages 299–313, 1996.

K. Shoemake. Animating rotation with quaternion curves. *ACM SIGGRAPH Computer Graphics*, 19(3):245–254, July 1985. doi: 10.1145/325165.325242.

G. Simmons and Y. Demiris. Optimal robot arm control using the minimum variance model. *Journal of Robotic Systems*, 22(11):677–690, 2005. doi: 10.1002/rob.200092.

A. Skoglund, B. Iliev, B. Kadmiry, and R. Palm. Programming by demonstration of pick-and-place tasks for industrial manipulators using task primitives. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 368–373, Jacksonville, Florida, June 20-23 2007. doi: 10.1109/CIRA.2007.382863.

A. Skoglund, T. Duckett, B. Iliev, A. Lilienthal, and R. Palm. Programming by demonstration of robotic manipulators in non-stationary environments. In *Proceedings of 2006 IEEE International Conference on Robotics and Automation*, 2006.

A. Skoglund, B. Iliev, and R. Palm. A hand state approach to imitation with a next-state-planner for industrial manipulators. In *Proceedings of the 2008 International Conference on Cognitive Systems*, pages 130–137, University of Karlsruhe, Karlsruhe, Germany, April 2-4 2008.

A. Skoglund, J. Tegin, B. Iliev, and R. Palm. Programming-by-demonstration of reach to grasp tasks in hand-state space. In *Submitted to the 14:th International Conference on Advanced Robotics*, Munich, Germany, June 22-26 2009.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

R. S. Sutton. Dyna, an Integrated Architecture for Learning, Planning, and Reacting. In *SIGART Bulletin*, volume 2, pages 160–163, 1991.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: an Introduction*. The MIT Press, 1998.

T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(1):116–132, January/February 1985.

J. Tani, M. Ito, and Y. Sugita. Self-organization of distributedly represented multiple behavior schemata in mirror systems: Reviews of robot experiments using rnnpb. *Neural Networks*, 17(8–9):1273–1289, 2004. doi: 10.1016/j.neunet.2004.05.007.

J. Tegin, S. Ekvall, D. Kragic, B. Iliev, and J. Wikander. Demonstration based learning and control for automatic grasping. In *International Conference on Advanced Robotics*, Jeju, Korea, Aug 2007.

J. Tegin, J. Wikander, and B. Iliev. A sub €1000 robot hand for grasping – design, simulation and evaluation. In *International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, Coimbra, Portugal, Sep 2008.

J. Tegin, S. Ekvall, D. Kragic, J. Wikander, and B. Iliev. Demonstration based learning and control for automatic grasping. *Journal of Intelligent Service Robotics*, 2(1):23–30, 2009. doi: 10.1007/s11370-008-0026-3.

A. Ude. Trajectory generation from noisy positions of object features for teaching robot paths. *Robotics and Autonomous Systems*, 11(2):113–127, 1993.

S. Vijayakumar and S. Schaal. Locally weighted projection regression: An *O(n)* algorithm for incremental real time learning in high dimensional space. In *Proceeding of Seventh International Conference on Machine Learning*, pages 1079–1086, 2000.

S. Vijayakumar, A. D'Souza, and S. Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634, 2005. doi: 10.1162/089976605774320557.

C. C. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

M. Weigelt, R. Cohen, and D. A. Rosenbaum. Returning home: location memory versus posture memory in object manipulation. *Experimental Brain Research*, 179(2):191–198, 2007. doi: 10.1007/s00221-006-0780-4.

J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, Jan 2001. doi: 10.1126/science.291.5504.599.

D. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7–8):1317–1329, 1998.

D. M. Wolpert and Z. Ghahramani. Computational principles of movement neuroscience. *Nature Neuroscience*, 3(11):1212–1218, Nov 2000.

D. M. Wolpert, R. C. Miall, and M. Kawato. Internal models in the cerebellum. *Trends in Cognitive Sciences*, 2(9):338–347, 1998.

K. Yamane and Y. Nakamura. Dynamics filter - concept and implementation of online motion generator for human figures. *IEEE Transactions on Robotics and Automation*, 19(3):421–432, 2003. doi: 10.1109/TRA.2003.810579.

T. R. Zentall. Imitation: definitions, evidence, and mechanisms. *Animal Cognition*, 9(4):335–353, 2007. doi: 10.1007/s10071-006-0039-2.

K. J. Åström and T. Hägglund. *PID Controllers: Theory, Design and Tuning*. The International Society for Measurement and Control, second edition, 1995.

Publications *in the series*
Örebro Studies in Technology

1. Bergsten, Pontus (2001) *Observers and Controllers for Tagaki-Sugeno Fuzzy Systems*. Doctoral Dissertation.
2. Iliev, Bokyo (2002) *Minimum-Time Sliding Mode Control of Robot Manipulators*. Licentiate Thesis.
3. Spännar, Jan (2002) *Grey Box Modelling for Temperature Estimation*. Licentiate Thesis.
4. Persson, Martin (2002) *A Simulation Environment for Visual Servoing*. Licentiate Thesis.
5. Boustedt, Katarina (2002) *Flip Chip for High Volume and Low Cost – Materials and Production Technology*. Licentiate Thesis.
6. Biel, Lena (2002) *Modeling of Perceptual Systems – A Sensor Fusion Model with Active Perception*. Licentiate Thesis.
7. Otterskog, Magnus (2002) *Produktionstest av mobiltelefonantenner i mod-växlande kammare*. Licentiate Thesis.
8. Tolt, Gustav (2004) *Fuzzy-Similarity-Based Low-level Image Processing*. Licentiate Thesis.
9. Loutfi, Amy (2003) *Communicating Perceptions: Grounding Symbols to Artificial Olfactory Signals*. Licentiate Thesis.
10. Iliev, Boyko (2004) *Minimum-time Sliding Mode Control of Robot Manipulators*. Doctoral Dissertation.
11. Pettersson, Ola (2004) *Model-Free Execution Monitoring in Behavior-Based Mobile Robotics*. Doctoral Dissertation.
12. Överstam, Henrik (2004) *The Interdependence of Plastic Behaviour and Final Properties of Steel Wire, Analysed by the Finite Element Method*. Doctoral Dissertation.
13. Jennergren, Lars (2004) *Flexible Assembly of Ready-to-Eat Meals*. Licentiate Thesis.
14. Li, Jun (2004) *Towards Online Learning of Reactive Behaviors in Mobile Robotics*. Licentiate Thesis.
15. Lindquist, Malin (2004) *Electronic Tongue for Water Quality Assessment*. Licentiate Thesis.
16. Wasik, Zbigniew (2005) *A Behavior-Based Control System for Mobile Manipulation*. Doctoral Dissertation.

17. Berntsson, Tomas (2005) *Replacement of Lead Baths with Environment Friendly Alternative Heat Treatment Processes in Steel Wire Production*. Licentiate Thesis.
18. Tolt, Gustav (2005) *Fuzzy Similarity-based Image Processing*. Doctoral Dissertation.
19. Munkevik, Per (2005) *Artificial sensory evaluation – appearance-based analysis of ready meals*. Licentiate Thesis.
20. Buschka, Par (2005) *An Investigation of Hybrid Maps for Mobile Robots*. Doctoral Dissertation.
21. Loutfi, Amy (2006) *Odour Recognition using Electronic Noses in Robotic and Intelligent Systems*. Doctoral Dissartation 2006.
22. Gillström, Peter (2006) *Alternatives to Pickling; Preparation of Carbon and Low Alloyed Steel Wire Rod*. Doctoral Dissertation.
23. Li, Jun (2006) *Learning Reactive Behaviors with Constructive Neural Networks in Mobile Robotics*. Doctoral Dissertation.
24. Otterskog, Magnus (2006) *Propagation Environment Modeling Using Scattered Field Chamber*. Doctoral Dissertation.
25. Lindquist, Malin (2007) *Electronic Tongue for Water Quality Assessment*. Doctoral Dissertation.
26. Cielniak, Grzegorz (2007) *People Tracking by Mobile Robots using Thermal and Colour Vision*. Doctoral Dissertation.
27. Boustedt, Katarina (2007) *Flip Chip for High Frequency Applications – Materials Aspects*. Doctoral Dissertation.
28. Soron, Mikael (2007) *Robot System for Flexible 3D Friction Stir Welding*. Doctoral Dissertation.
29. Larsson, Sören (2007) *An industrial robot as carrier of a laser profile scanner – Motion control, data capturing and path planning*. Doctoral Dissertation.
30. Persson, Martin (2008) *Semantic Mapping Using Virtual Sensors and Fusion of Aerial Images with Sensor Data from Ground Vehicle*. Doctoral Dissertation.
31. Andreasson, Henrik (2008) *Local Visual Features base Localisation and Mapping by Mobile Robots*. Doctoral Dissertation.
32. Bouguerra, Abdelbaki (2008) *Robust Execution of Robot Task-Plans: A Knowledge-based Approach*. Doctoral Dissertation.
33. Lundh, Robert (2009) *Robots that Helps Each Other: Self-Configuration of Distributed Robot Systems*. Doctoral Dissertation.

34. Skoglund, Alexander (2009) *Programming by Demonstration of Robot Manipulators*. Doctoral Dissertation.