Imitation Learning of Motor Skills for Synthetic Humanoids

Von der Fakultät für Mathematik und Informatik

der Technischen Universität Bergakademie Freiberg
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades

Doktor-Ingenieur

Dr.-Ing.

vorgelegt

von Dipl.-Inform. Heni Ben Amor

geboren am 01.02.1982 in Dernbach

Gutachter:
Prof. Dr.-Ing. Bernhard Jung, Freiberg
Prof. Dr. rer. nat. Ulrich Furbach, Koblenz

Tag der Verleihung: 12.11.2010

Versicherung

Hiermit versichere ich, daß ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich keine Unterstützungsleistungen Dritter, wie Promotionsberater, in Anspruch genommen. Weitere Personen haben von mir keine geldwerten Leistungen für Arbeiten erhalten, die nicht als solche kenntlich gemacht worden sind.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Freiberg, den 01. Juni 2010

Heni Ben Amor

Acknowledgements

You can dream, create, design and build
the most wonderful place in the world, but
it requires people to make the dream a reality.

(Walt Disney)

Although the focus of this thesis are virtual and robotic humanoids, it is the interaction with humans of flesh and blood that played the most important part in its completion. During the last four years, many people at the Technical University Bergakademie Freiberg and the University of Osaka supported me in the work leading to this thesis.

First, I would like to thank my advisor Bernhard Jung for giving me the opportunity to work on exciting scientific projects, and for guiding and educating me during my years in Freiberg. Bernhard is a very inspiring person who always managed to create a motivating atmosphere in our working group. Thank you Bernhard for all the ideas, your help and patience! I would also like to thank my coworkers at the virtual reality group Guido Heumer, Arnd Vitzthum and Matthias Weber for the stimulating scientific (and sometimes non-scientific) discussions and their great support.

I am grateful to Ulrich Furbach for initiating me to the world of artificial intelligence and for being available as a reviewer of this thesis.

I am indebted to Hiroshi Ishiguro for making me part of the Intelligent Robotics Lab at the University of Osaka in Japan. Ishiguro-sensei was a great mentor and constantly supported me in my research on android robots.

My special thanks go to Konrad Froitzheim for the numerous hints on how to improve the thesis and for his support in acquiring the small humanoid robots that were essential for carrying out the experiments presented here.

I would like to thank my close collaborator Shuhei Ikemoto for his constant help as well as his support during my stays in Japan. Although at times we were seperated by several thousands of kilometers, we always managed to continue our successful collaboration. I had a great time working with Shuhei and I will always remember the funny discussions at 3:00 a.m. in the morning on the way home from the lab.

I thank Oliver Obst for introducing me to robotics, machine learning, scientific writing and many other things that were important for my later career.

I am deeply thankful to all of my students who supported and helped me in various ways, especially Erik Berger, David Vogt, Henry Lehmann, Edith Kegel, Christian Schlegel, Ralf Müller, Maik Deininger, Stefanie Höfig, Peter Scheicher and Eric Kunze.

Without the delicious couscous of Moncef Bouaziz, his constant support, as well as his great humor, the writing of this thesis would have taken a significantly longer time.

A big *Thank You!* also goes to Martin Wiesenmayer for proofreading this thesis and for keeping me up-to-date about recent cinematic activities.

I thank my sisters Hazar and Houyem for being an endless source of inspiration and for cheering me up during the hard times of solitary writing and experimentation.

Finally, I want to thank my brother Heikel, my sister Hounaida and my parents Hedi and Mesaouda Ben Amor for their moral support and for always being there when I need them.

Freiberg, 2010

Contents

Mathematical Notation

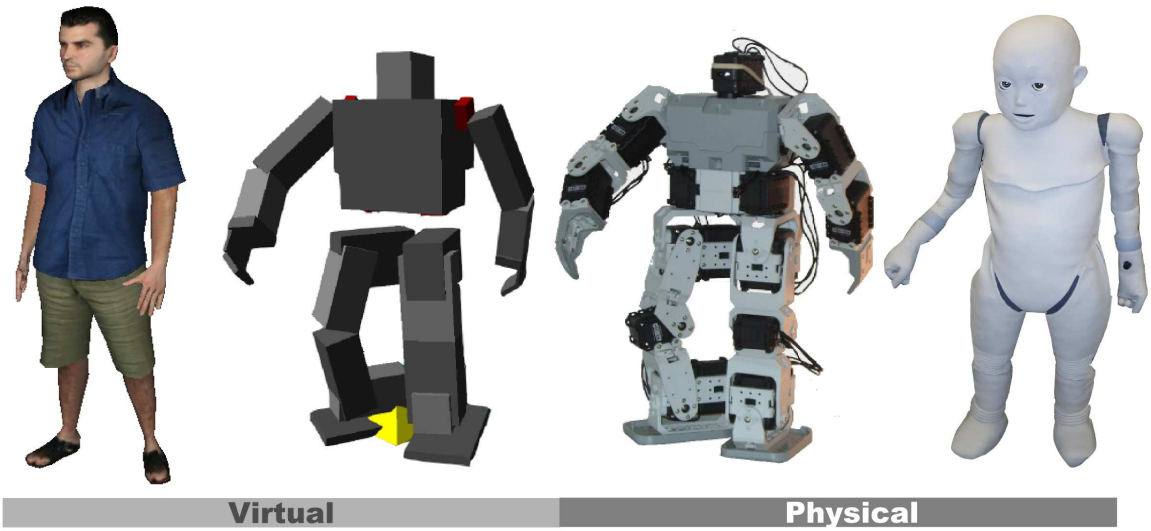
E	Error function.
$\mathbb{P}$	Low-dimensional posture space.
Θ	Calculates the fingertip position.
$\blacktriangle(s_1, s_2, s_3)$	Polygon spanned between three points $s_1, .., s_3$.
$\mathcal{D}(\boldsymbol{x}(i), \boldsymbol{x}(j))$	Calculates Euclidean distance between two vectors.
F	Fitness function used for optimization.
H	Dimensionality of input data.
K	Number of nearest neighbors.
L	Dimensionality of projected data.
Ψ	Mapping from high-dimensional space $\mathbb{R}^H$ to $\mathbb{P}$.
W	A matrix.
W_{ij}	An entry of matrix W.
X	The data set in matrix notation.
Σ	Covariance matrix.
$\mathcal{H}_{\beta(t)}$	Heaviside step function.
$\nabla f(x)$	Derivative of function f in x.
N	Number of points in the data set.
Φ	Probability density function.
$\mathcal{M}$	A manifold (noted as a set).
$\mathcal{X}$	The input data set.
$\widetilde{\mathcal{X}}$	The set of projected data points.
$\boldsymbol{\mu}$	Mean of a data cloud.
$\widetilde{\boldsymbol{x}}$	Vector in a low-dimensional space.
$\widehat{\boldsymbol{x}}$	Reprojected vector.
$\boldsymbol{x}(i)$	The ith vector of data set $\mathcal{X}$.
$V_B(m)$	Violation function.
$\mathcal{N}(\boldsymbol{x}\|\boldsymbol{\mu}, \Sigma)$	Normal distribution with mean $\boldsymbol{\mu}$ and covariance Σ.

Acronyms

ANN	Artificial neural network.
BFGS	Broyden-Fletscher-Goldfarb-Shanno.
BIC	Bayesian information criterion.
CCA	Curvilinear component analysis.
CDA	Curvilinear distance analysis.
CNO	Contact normal opposition.
COF	Cone of friction.
DHC	Dynamic hill-climbing.
DIP	Distal-interphalangeal.
DMP	Dynamic motor primitives.
DOF	Degrees of freedom.
DR	Dimensionality reduction.
EA	Evolutionary algorithm.
EES	Encapsulated evolution strategies.
EM	Expectation-maximization.
ES	Evolution strategies.
EVD	Eigenvalue decomposition.
GCS	Grasp coordinate system.
GMM	Gaussian mixture model.
GMR	Gaussian mixture regression.
HMM	Hidden Markov model.
IK	Inverse kinematics.
Isomap	Isometric feature mapping.

LLE	Locally linear embedding.
MCP	Metacarpal-phalangeal.
MDS	Multidimensional scaling.
NLDR	Nonlinear dimensionality reduction.
NLM	Sammon's nonlinear mapping.
PbD	Programming by demonstration.
PCA	Principal component analysis.
PHRI	Physical human-robot interaction.
PIP	Proximal-interphalangeal.
PLDPM	Probabilistic low-dimensional posture model.
RPROP	Resilient backpropagation.
SGI	Stability grasp index.
SOM	Kohonen's self-organizing map.

1. Introduction



The evolution of interfaces to computers and other devices resulted in the development of a range of synthetic humanoids—artifacts with human-like appearance and behavior—that are used in various fields, including research, entertainment and training. Some of these synthetic humanoids exist in virtual space, like digital actors in movie productions or avatars replacing human users in online games. In contrast to that, other synthetic humanoids, such as humanoid and android robots, have a physical manifestation within our real world. Yet, regardless of whether they exist in a virtual or real space, synthetic humanoids need to move and behave in a realistic way in their environment in order to appear lifelike.

So far, the generation of such motions is typically left to skilled experts. For example, in most movie productions and video games animators manually set and modify important postures, the so-called *keyframes*. In order to produce realistic movements with this approach, they need to be well-trained in traditional animation techniques. In robotics, motions are often specified in terms of computer programs written by expert programmers. In both cases the process of motion specification is labour intensive, time-consuming and limited to a small number of experts. At the same time, synthetic humanoids are faced with more and more complex application scenarios. In recent years, virtual characters are increasingly used in environments with user-created content in which they have to respond to unknown situations. Similarly, the application domain

of robots is steadily expanding from well-defined environments in research labs and manufacturing plants, to complex, cluttered environments like homes or schools. In order to cope with these challenges, we need flexible motion production methods that meet the following two requirements:

1. an easy and intuitive way of specifying new motions
2. the ability to adapt motions to the current context

The goal are synthetic humanoids whose repertoire of motor skills can easily be expanded. In addition, the motion production algorithm must be able to synthesize variants of an acquired skill in order to cope with changes in the environment or changes in the anatomical properties. For example, once a synthetic humanoid has learned to grasp an object, it should be able to reproduce the skill even if the position and/or orientation of the object is changed.

1.1. Motivation

In the last twenty years, the use of synthetic human-like agents has become increasingly popular. In particular the entertainment and gaming industry proved to be a catalyst for this development. The depiction of humans in games moved from simple pixel-based sprites to nearly photo-realistic 3D models. At the same time, many other industry branches adopted the use of synthetic humanoids for their purpose. Today, synthetic humanoids are used in a wide range of applications including:

- simulation of pedestrian movement and evacuation scenarios
- ergonomic assessment and virtual prototyping of products
- simulation and visualization of stunts in movie productions
- virtual meeting rooms for remote meetings of business partners

Until recently, humanoid robots mostly remained the realm of research, yet, in recent years the technology behind these robots significantly progressed and matured. Early research was mostly concerned with walking skills for legged mechanical devices. Today, scientists are turning towards android robots whose visual realism has reached a level, where they often cannot easily be distinguished from real persons anymore. The RoboCup research community is even working towards: “a team of fully autonomous humanoid robots that can play and win against the human world champion soccer team” by the year 2050 [KAK+97].

The automotive industry has shown strong interest in humanoid and android robots. Today all major Japanese car companies have efforts dedicated to the development of humanoid robots. According to the estimations of the International Federation of Robotics

(IFR) [IFoR10], about 4.4 million domestic robots and 2.8 million entertainment robots have been sold by the end of the year 2009. So far, only a fraction of these robots has a humanoid appearance. However, the IFR reports note that humanoid robots are an application area with strong growth potential. They also predicted that many countries will use humanoid robots to provide everyday services in the public sector. Japan is at the forefront of this development. Partly motivated by an ageing society, Japanese researchers are working towards the use of robots for the assistance of humans in everyday tasks. The city of Osaka is already working on a large facility in central Osaka—the RobotCity Core—which opens in 2011 and will contain open laboratories where researchers, robots and consumers can interact.

As evident from the examples above, the application domains of synthetic humanoids keep steadily expanding. On the other hand, making them move in an intelligent way in their environment remains a complex and time-consuming process. This thesis contributes to the solution of these problems by providing new methods that simplify the motion generation process while at the same time improving the robustness of the resulting motor skills with regard to changes in the environment. Our work is part of an international research effort dedicated to this problem. More precisely, the work presented here was carried out within two research projects, the Virtual Workers project (Germany, website: `http://vr.tu-freiberg.de/virtualworkers/`) and the JST ERATO Asada Synergistic Intelligence Project (Japan, website: `http://www.jeap.org/`). The Virtual Workers project aims at developing autonomous virtual humans for prototyping scenarios, whereas the Synergistic Intelligence project studies new types of interactions between robots, humans and environments.

1.2. Problem Statement

The main problem adressed in this thesis can be stated as follows:

How can we teach dynamic motor skills in a natural and intuitive way to synthetic humanoids, such that a learned skill can later be reproduced in variable situations and environments?

The following sub-questions arise from this problem statement:

- How can we teach motor skills with a minimum of programming effort?
- Is it possible to create a general approach for teaching motor skills that supports both virtual humans and different types of physical robots?
- How should human motion and posture be represented in order to support this process?
- How can trained motions be adapted to new situations?

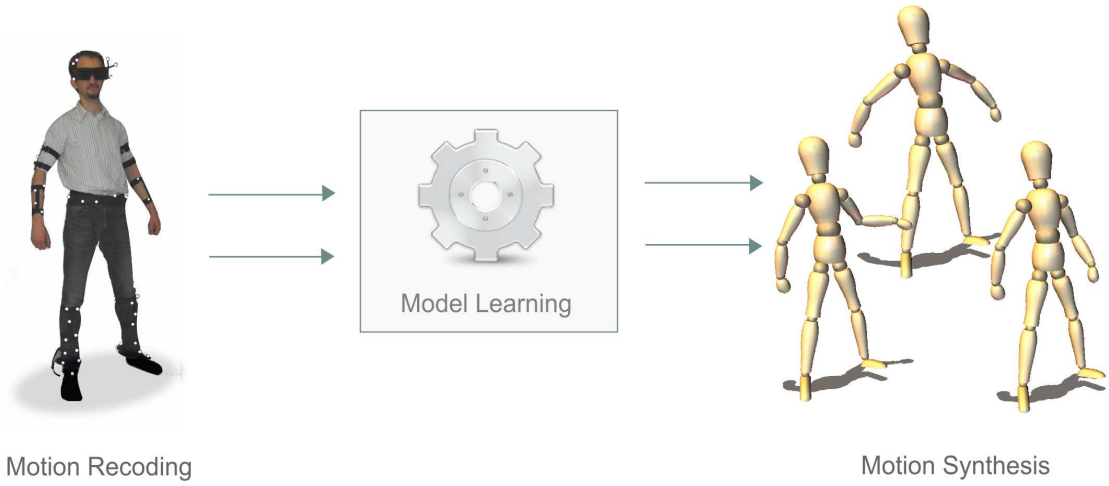


Figure 1.1: The imitation learning approach used in this thesis: Demonstrations of the skill are recorded from a human demonstrator using motion capture or other tracking devices. The recorded motions are then processed and a statistical model thereof is learned. In the final step, the model is used to synthesize variations of the demonstrated skill. The synthesis process takes into account the current situation.

1.3. Methodology

This thesis presents a general approach for teaching motor skills to synthetic humanoids that lies at the intersection of artificial intelligence, robotics, computer animation and human-machine interfaces. Our approach is inspired by imitation in natural beings. Humans are able to learn even complex motions by observing and reproducing the behavior of other humans. Through the means of imitation we can incrementally expand our repertoire of skills.

The approach presented in this thesis follows the same idea. First, a demonstrator presents one or several examples of the skill to be learned (see Figure 1.1). The demonstrations are recorded using modern sensor technology, such as motion capture systems or robot sensors. However, in contrast to most other animation approaches the resulting data is *not* simply stored in a database. Instead, it is used to derive models of the motion at hand. The goal of this process is to automatically extract as much information as possible about the demonstrated behavior, which in turn can be used to synthesize new motions. To this end, we will propose a novel type of statistical models which is particularly well suited for the compact representation of human motion. Dimensionality reduction and machine learning algorithms lie at the core of this model.

Once a statistical model is learned, it can be used to generate a large number of variations of the demonstrated motion. This allows us to search within the learned statistical models for motions that best fit the current situation. Hence, even if the environment changes, we can still synthesize meaningful motions. For example, if the position of an

object changes after the recording of a grasping motion, a modified animation can be synthesized so that the object is still correctly grasped. We will show that the approach can be used at different scales of complexity including single finger movement, hand movement and full-body movement. Furthermore, we will see that the approach can successfully be used to perform different types of adaptations ranging from adaptations to geometric constraints to the adaptation to a human interaction partner.

Our approach tries to bridge the gap between different disciplines. An important goal in this regard is to ensure that the proposed methodology is applicable to different types of synthetic humanoids. With this goal in mind, we will use the proposed approach in scenarios that involve virtual humans, small, commercially available humanoid robots, as well as state-of-the-art android robots.

Motor skills that are learned using this approach can be regarded as building blocks for complex sequences of motions. Learned behaviors such as walking, grasping or pushing can be chained together to produce sequences of arbitrary length. We will present a new framework that enables the user to specify sequences of behaviors at a high level of abstraction. Using this framework, a synthetic humanoid can be animated from textual descriptions. The user specifies the behaviors to be executed as well as their timing and the objects involved using an XML document. The document is then processed by the framework and translated into an animation. Changes to the animation can simply be performed by changing the original text document on which it is based. For example, the user can change the object which is manipulated.

1.4. Contributions

This thesis makes five contributions to the control and animation of synthetic humanoids:

Models for Imitation Learning of Motor Skills. Probabilistic low-dimensional posture models (PLDPM) are compact mathematical models of human motion that can be learned from recorded example data. They are used to synthesize realistic variations of a learned motor skill. PLDPMs lie at the core of a general imitation learning approach that can be used to synthesize context-dependent motions for a wide range of synthetic humanoids.

Data-driven Grasp Synthesis Algorithm. We present a data-driven animation method for the synthesis of natural looking human grasping. Data acquired through motion capture is used to learn mathematical models for different types of human grasps. These models are used to rapidly synthesize realistic grasps on new graphical objects.

Robot Motion Programming through Kinesthetic Bootstrapping. Kinesthetic bootstrapping is a novel touch-based programming technique that allows users to teach dynamic, full-body motions to humanoid robots. The human instructs the robot by

manually moving its joints and body to postures that approximate the intended movement. The movement is then recorded and further optimized within a physics based virtual environment.

Learning Algorithm for Physical Human-Robot Interaction. Physical interaction learning is an online learning algorithm which aims at improving the interaction between humans and humanoid robots in settings that involve close physical contact. The algorithm uses human critique in order to adapt the behavior of the robot. In experiments with a state-of-the-art android we show that the proposed algorithm significantly improves the joint coordination during cooperative tasks. To the best of our knowledge, these experiments are among the first experiments worldwide to feature close-contact interaction between a human and a learning android robot.

Imitation of Action Sequences for Virtual Prototyping. Building upon the action capture [JAHV10] technique, we present an animation framework for virtual humans. The framework allows us to synthesize sequences of actions from textual descriptions. These textual descriptions can be either hand-crafted or recorded from human interactions in an immersive virtual environment. The framework is especially well suited for virtual prototyping scenarios.

1.5. Thesis Outline

Chapter 2 reviews related work on imitation learning of motor skills. Our survey starts with work on imitative capabilities in humans and animals, then proceeds to imitation learning in the area of robotics, and finally reviews related techniques in the field of computer animation.

Chapter 3 presents the mathematical background on articulated structures and dimensionality reduction. Articulated structures, and in particular kinematic chains, are important for modeling moving human-like characters. Dimensionality reduction on the other hand is an essential part of the imitation learning approach presented in this thesis.

Chapter 4 defines a new, general approach for imitation learning of motor skills by synthetic humanoids. For this, the concept of *probabilistic low-dimensional posture models* will be introduced and formalized. Additionally, optimization theory and actual optimization algorithms will be introduced. Based on a simple example, the imitation learning approach, as well as the specific steps involved are explained in detail.

Chapter 5 presents an algorithm for synthesis of natural human grasping that is based on the proposed imitation learning approach. The chapter introduces basic anatomical features of the human hand, different classification schemes for human grasping, as well as metrics for assessing the quality of a given grasp. The second part of this chapter describes our grasp synthesis algorithm as well as approaches to reduce the computation times. The algorithm is then evaluated in a set of incremental experiments.

Chapter 6 expands the imitation learning approach to motions involving the entire body of a synthetic humanoid. We show how locomotion skills of virtual humans are learned from motion capture data. Furthermore, we present novel methods for programming robots by tactile demonstration and by close-contact interactions.

Chapter 7 presents a framework for imitation of sequences of actions. First, in this chapter we briefly describe the action capture method and how human interactions can be recorded in a virtual environment. Then, we will present a set of behaviors which can be combined to generate animations of action sequences. These behaviors cover a wide range of manipulation skills. Finally, we show several virtual prototyping scenarios in which the described behaviors are used to imitate long sequences of actions.

Chapter 8 concludes this thesis and outlines the most promising directions for future work.

1.6. Overview of the Main Chapters

Figure 1.2 gives an overview of the main chapters of this thesis. Chapter 4 is the core chapter of the thesis and presents the central idea and formalization of the proposed learning method. For a better understanding of the involved steps, we present a simple example for the imitation of motions for a single finger. With each consecutive chapter, the scope of application then steadily expands. Chapter 5 presents the application of the proposed method for the synthesis of motions and postures of a full hand.

In Chapter 6 three different scenarios are presented, which all deal with motions involving the entire body of different types of synthetic humanoids. Section 6.2, deals with imitation of motions recorded via motion capture. In Section 6.3 we present an application of the method for programming humanoid robots by demonstration. The demonstrations are not recorded via motion capture anymore. Instead, they are acquired from touch interactions. Section 6.4 further extends this scenario and realizes learning within close-contact interaction between a human caregiver and the robot. Here, we will focus on adapting the *timing* of an android robot's motions, so that interaction with a human caregiver can be successful. Up to this point, single behaviors are learned and added to the repertoire of the imitating agent.

In Chapter 7 we will finally show how sequences of behaviors can be imitated. The action capture method which we will present in this regard, builds on the imitation learning approach from Chapter 4 and extends it through an abstract representation of behavior sequences.

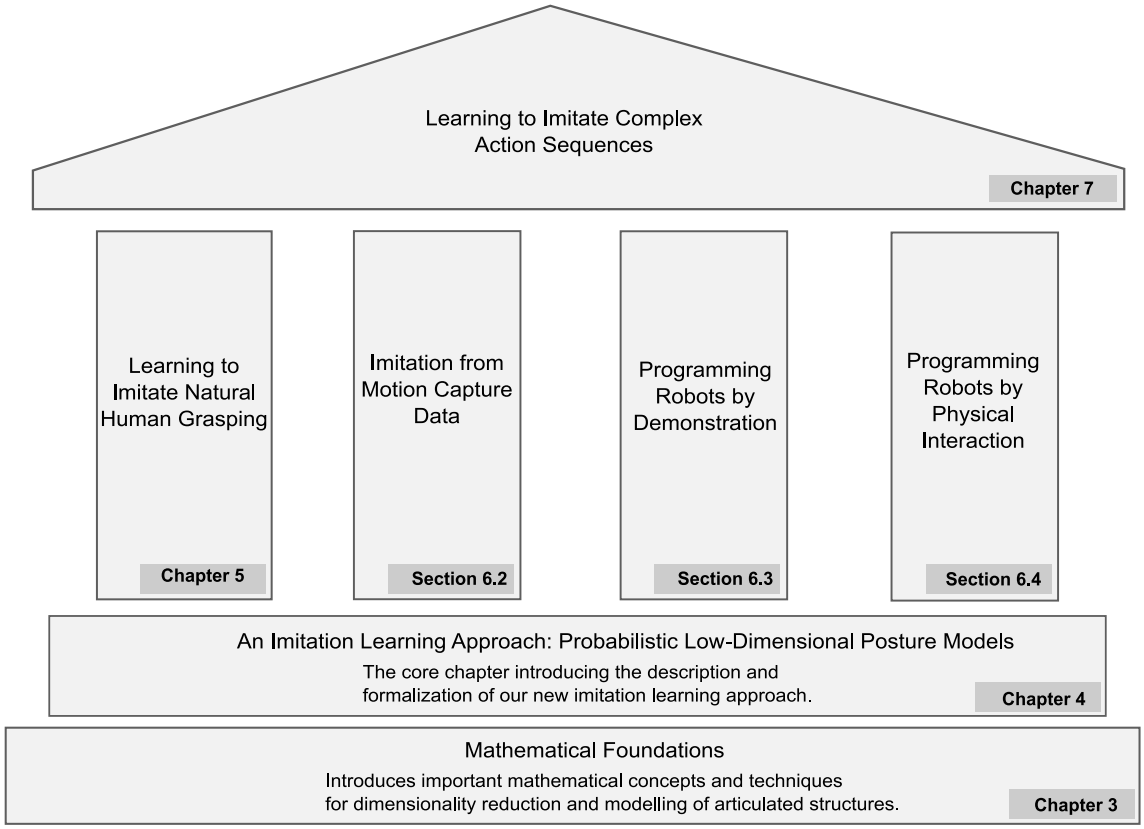


Figure 1.2: Overview over the main chapters.

2. Related Work on Imitation Learning

In this chapter, we present prior work on imitation learning from the domains of biology, robotics and computer animation. Drawing from the analysis of imitative abilities of humans and animals, we turn to the state-of-the-art in computational models of imitation. In particular, we discuss the use of imitation learning for teaching new skills to robots. Next, we present current work in computer animation that, although not directly linked to imitation, deals with variation and adaptation of recorded movements. Finally, we explain how the work presented in this thesis draws upon previous research and extends previous approaches using concepts of human imitation learning.

2.1. Imitation in Humans and Animals

According to Thorndike [Tho98], imitation is: "From an act witnessed learn to do an act". Although, intuitively clear, there is to our day disagreement about its exact interpretation as well as about the species capable of this ability. According to recent research results, it is safe to assume that dolphins [Her02], pigeons [ZSS96] and humans [MM77; MM97] are endowed with imitative capabilities. Yet, in the literature we find many different terms describing manifestations of imitation, including *observational learning*, *response facilitation*, *goal emulation* or *stimulus enhancement*. We will concentrate on what is called *true imitation* [Tho63], i.e. the reproduction of a novel, previously unseen motor behavior. True imitation requires that the demonstrated behavior is not already part of the motor repertoire of the observer. Hence, reproduction of a newly seen tennis swing falls within the definition of true imitation, whereas mimicking another ones facial expressions does not. Additionally, it is required that the observer not only reproduces the goal of the seen behavior, but also the demonstrators means of achieving this goal, that is, the same movements.

Imitation in behavioral sciences Imitation of movements has been a major research topic in behavioral sciences. Meltzhoff and coworkers [MM77] showed that imitation already takes place in newly born children. Their observations led to the four stage model of progression of imitative abilities [Mel96]. In the first stage, a so-called body babbling phase, the infant explores its body and learns how specific muscle movements achieve elementary body configurations. The result is an internal model of the infant's own body. In the next phase, the infant uses its body parts to imitate observed body movements. For this, it first activates the corresponding body part, then corrects

the imitative response until convergence on an accurate match. In the third stage, the infants learn to imitate manipulations on objects. This includes playing with toys in a variety of contexts. In this stage, the imitative behavior becomes more goal oriented. Although the child might not infer the purpose of the manipulation task, it is able to understand the basic steps involved and reproduce the task under different conditions. Such a generalization of seen behavior needs more sophisticated internal models, in which relations between actions and objects are stored, too. Finally, in the last phase, infants are able to understand seen actions and infer goals and intentions. Generally, imitation involves some kind of *adaptation* or generalization, that is, the ability of adapting a learned behavior to a new situation or environment. This entails the ability to produce many *variations* of a seen movement. For example, a learned manipulation movement can be adapted to fit the shape or location of a new object. We will see in Section 4, that adaptation and variation are also the basic concepts for imitation in synthetic humanoids.

Neurophysiological basis of imitation The neurophysiological basis for imitative capabilities, the so-called mirror neurons [DFF+92; RFFG96; GFFR96], was discovered towards the end of the 20th century. It was observed that many neurons in the ventral F5 area of a macaque monkey's brain showed activity both during execution and observation of an action. In the study, single cells in the brain of the monkey both fired when the monkey performed a grasp and when it observed a grasp being executed by the experimenter or another monkey. This was seen as an evidence for overlapping neural substrate for action execution and observation in humans as well as other primates. In other words, these neurons mirrored the observed behavior onto the motor system. Because of this behavior they were dubbed *'mirror neurons'*. The exact role of mirror neurons and mirror neuron systems is still an issue of debate. Yet, there is evidence that links mirror neurons to action understanding, empathy and imitation. In particular, the mirror neuron system is often considered to be the neural basis for imitation learning. In [ABIO00] Arbib and colleagues hypothesize: "mirror systems orchestrate the various components involved in the sensorimotor transformations required by imitation".

Computational models of mirror neurons Several computational models for imitation learning have been proposed which are to different degrees based on the findings we discussed in the last two paragraphs. Billard presents a learning architecture that is loosely based on the neurological findings in primates in [Bil00]. It includes an abstract model of the spinal cord, primary motor cortex, the premotor cortex, the temporal cortex, and the cerebellum. The neurons in the neural network model of the premotor cortex respond to both visual observation of movements and to corresponding motor commands. Using different movement patterns it is shown that the model can be used to train virtual avatars by imitation. Another computational learning model that is based on the concept of mirror neurons was presented by Oztop [Ozt02]. Artificial neural networks and kinematic simulations are used to model the development of grasp learning in

infants. The results of these simulations indicate that self-executed, exploratory grasp actions during childhood can adapt parietal and premotor connections to shape mirror neurons. Triesch et al. [TJD07] presented a reinforcement learning algorithm which tries to reproduce the emergence of gaze following capabilities in human infants. The learned pre-motor representations exhibit many properties characteristic of mirror neurons. Other computational models for mirror systems can be found in [WW01], [BL09] and [TIS04]. The mirror neuron theory provides an interesting view on action recognition and generation in humans and animals. Yet, there are many opposing views on the subject and many early hypotheses on the role of these neurons remain controversial. Most notably, it is questioned, whether mirror neurons are an important requisite of imitation. Therefore, most published computational models based on mirror neurons are used to generate simulation results that are in favor of or that challenge existing theories.

Consequently, the mirror system is still not sufficiently understood to be used as a basis for computational approaches to imitation learning. Existing computational realizations focus on specific phenomena, such as the reproduction of simple arm oscillatory movements, and do not scale up to more complex movements. Additionally, to the best of our knowledge, there is only little work investigating how imitated behaviors can be adapted to new environments or situations. In contrast to that, more promising and general computational approaches to imitation learning have been investigated in the context of programming by demonstration, which we discuss in the following section.

2.2. Programming by Demonstration

Computational approaches to imitation learning have received increasing attention in robotics. Often referred to as Programming by demonstration (PbD), imitation learning is viewed as: "a promising route to automate the tedious manual programming of robots" [Sch99]. In PbD a user does not specify the robot's movements using traditional programming languages. Instead, he only provides a demonstration of the desired behavior. Based on this example, the robot autonomously generates a control program, allowing it to reproduce the skill in different situations.

According to Billard et al. [BCDS08], there are three reasons for the ongoing interest in computational approaches to imitation learning. First of all, imitation learning helps to reduce the search space for learning. The provided examples can be used to bootstrap learning algorithms and to focus on a small set of potential solutions. Second, imitation learning provides a more natural and user-friendly way of programming robots. It does not require any expert knowledge, as most people are already familiar with it from childhood and daily experience. Finally, by studying imitation in robots and artificial agents, we may learn more about the same processes in natural beings.

Early approaches to Programming by Demonstration First attempts to robot imitation learning were realized in the late 1970s and the early 1980s in the context of manufacturing robots. These approaches were termed *teach-in*, *guiding* or *play-back* and had low generalization capabilities [LP83]. Typically, a user manually moves a robotic effector to locations important for achieving the task and selects from a given set of executable operations, e.g. closing the gripper. Meanwhile, the robot records all internal joint coordinates as well as the executed operations. At the end of this process, the robot is able to generate an exact reproduction of the demonstrated actions by executing the recorded data in sequence. While this approach makes robot programming accessible to lay people and allows for precise reproduction of motion trajectories by the robot, it still suffers from major limitations. In particular, it is not possible to alter the recorded sequence of operations and movements later on. Consequently, the robot is not able to respond to changes in the environment at execution time. Later approaches to this problem try to synthesize robot control programs from the recorded low-level data. This is done by segmenting the trajectories into sequences of basic operations such as *grasp-object*, *move-to* or *close-to*. Both actions as well as spatial relationships between actions are represented in a symbolic way. Symbolic reasoning and planning are then used to adapt an already learned action sequence to a new context. Many of the early PbD systems based on symbolic representations provide highly task-specific solutions. In [KII94], for example, a system for the imitation of block assembly tasks is introduced. In a similar vein, Kang and Ikeuchi developed a system for automatic replication of grasps [KI95]. A data glove is used to record and classify the grasps to be reproduced. While suitable for assembly robots, these early approaches to PbD did not account for learning complex dynamic or ballistic movements. The focus was on the reproduction of the right sequence of predefined actions, rather than reproducing complex new movements.

Numerical approaches to Programming by Demonstration In recent years, numerical approaches to PbD have received increasing attention. For example, Ijspeert and colleagues proposed a method based on dynamical systems [INS02] which was later termed dynamic motor primitives (DMP). DMPs are based on the following approach. First, the teacher's demonstrations are recorded as trajectories of the important limbs, e.g. hands and legs. Then dynamical systems and a memory-based regression method [CD88] are used to derive a compact numerical representation of the trajectories, a so-called *control policy*. In subsequent publications, they showed that this approach can be used to learn a variety of movement skills including tennis swings [INS02], locomotion or drumming [DSRI06].

Another way of encoding trajectories is to use statistical modeling methods. In the Mimesis Model [TYS+06], for example, a continuous hidden Markov model (HMM) is used for encoding example trajectories. It can later be used to generate a trajectory that is close to the example via the Viterbi algorithm. The model can also be used to recognize and classify trajectories. A similar approach to motion generation is presented by Calinon et al. [CGB07]. A set of example trajectories are encoded using Gaussian

mixture regression (GMR). Using GMR a smooth generalized version of the trajectory for the encoded skill can be extracted. Additionally, the task constraints can be modeled in a probabilistic fashion. As a result, the robot recognizes the parts of the task space where deviation from the generalized trajectory is allowed and in which parts the generalized trajectory must be reproduced exactly. For example, when picking up a chess piece, the robot can move his gripper in different ways to the object. However, at the end, the gripper has to be at a specific position in order to successfully pick up the chess piece. Using GMR such constraints can automatically be reconstructed from example trajectories. In later work (see [HGCB08]), this approach is combined with the dynamical models discussed earlier, in order to account for goal-directed motions. The target of the reaching motions is modeled as an attractor, guaranteeing that the system can adapt to displacements of the target's position. Another probabilistic approach to imitation learning is used in [AAGD08]. Common key points from demonstration trajectories were extracted and a corresponding HMM is learned. The approach is used to generate dual-arm object manipulations. However, the GMR model by [CGB07] is superior to similar HMM based approaches. This is mainly due to its continuous modeling of trajectories. HMM based approaches assume that the trajectories are discretized into a small set of control points, which results in discontinuities during generation. The method presented in this thesis is inspired by probabilistic techniques. In particular, the concept of modeling observed data with means of Gaussian mixtures will be adopted. However, we will not limit the imitative capabilities to trajectories only. Instead, we will show how above approaches can be extended to the modeling and reproduction of full-body motions.

Techniques based on artificial neural networks Other interesting approaches to PbD are based on artificial neural network (ANN). Ito and colleagues [IT04], for instance, use a recurrent neural network for teaching a humanoid robot cyclic movement patterns. The network shows a similar behavior as biological mirror systems, i.e. the neurons both fire when a movement pattern is observed and when it is generated. Artificial neural networks can learn complex, nonlinear mappings between sensory information and the desired response. For example, given a set of demonstration trajectories and the environmental settings in which they were triggered, an ANN can learn how to *produce* these trajectories as a function of the context. Accordingly, when the setting changes the ANN adapts to these changes and produces an appropriate output. In reference [TNNI08], recurrent neural networks are used for teaching a robot how to perform manipulations on toy objects. The system allows a human to interact and guide the robot during learning, in order to reduce the number of trials needed. In [TN03], Tatani and colleagues present a hierarchical ANN that performs a bidirectional mapping from a robot's control inputs to a low dimensional internal representation of learned motion patterns. By using a special network topology, the ANN can be forced to perform dimensionality reduction during learning. This feature is useful in order to control the robot using a low number of parameters. In [MMMI07] an ANN is used to map the

motions of a human demonstrator onto an android robot. The goal is to achieve realistic human-like motion including subtle features of biological movement in order to increase the believability of the robot. In an experiment, the ANN learned a mapping from the subject's current posture, recorded via motion capture, to the android's control inputs. The robot simultaneously replicated the movements of the human teacher. Differences in posture between the robot and the teacher were recorded by the motion capture system and consequently used for improving the mapping. However, the android did not develop any internal representations of the seen actions. In [WW01] it is shown that ANNs are usable to learn behaviors based on multimodal input. During observation the student robot receives the position and motor commands of the teaching robot as well as a language description of the task performed, e.g. *go* or *pick*. After learning, the student robot reproduces the seen actions from language instructions. Further, it is able to recognize and predict the teacher's behaviors.

Despite the remarkable results presented in the earlier publications and literature, ANNs still have several major drawbacks. First of all, ANNs can be difficult to parametrize. Parameters such as the appropriate network structure, learning rate, or neuron type need to be determined before learning. A wrong choice for these parameters can have a significant impact on the outcome of the learning process. This requires the user to have sufficient knowledge about theory and mechanics of neural systems, as well as the heuristics and the rules of thumb used in the community. Further, ANNs employ a distributed representation of knowledge with information disseminated among a set of neurons, which makes the analysis and introspection of their behavior very difficult. Debugging and error look up is time-consuming and inefficient. Finally, depending on the task at hand, neural network learning can take a considerable amount of time.

Imitation and dimensionality reduction Alternative approaches to PbD, that are becoming increasingly popular, rely on dimensionality reduction as a preprocessing step. For example in [CGGR07], principal component analysis is used in conjunction with ANNs, in order to teach a robot how to walk through imitation. For this, human walking gaits is first recorded using a motion capture system. The resulting data is then processed using PCA in order to generate a low-dimensional space of postures. Finally, an ANN is used to learn a control policy in the reduced space of postures. The work of Chalodhorn and colleagues mainly focuses on walking gaits and does not present a general approach for imitation of arbitrary motor skills. Additionally, the correspondence problem occurs—the problem of mapping observed movements to the imitator's body. In Chapter 6 we will discuss how this problem can be circumvented by using other teaching modalities. Another approach based on dimensionality reduction is presented in [JM03]. Movement primitives are automatically segmented from a given motion stream, using a spatio-temporal version of the Isomap algorithm [TdL00]. The resulting primitives are regarded as a vocabulary from which more complex motion sequences can be recomposed. In a similar manner, Thurau et al. [TBS04] use dimensionality reduction techniques to derive movement primitives for game characters. The movements of an

expert user are recorded and the corresponding motion streams are segmented into distinct elementary movements. During a game, the computer-controlled character uses probabilistic reasoning in order to decide which of the learned primitives to reproduce. Thurau et al. argue that this approach leads to the emergence of lifelike computer game characters exhibiting similar behavior as human players. An advantage of this approach is that the game characters steadily increase their playing skills. Their abilities and intelligence scale with the familiarity of user with the game. The better the human player becomes, the better become his enemy characters. Recently, DR techniques have also been combined with DMPs (see [BV09]). The promise of this approach is to reduce the computational complexity of the DMP algorithm, while at the same time providing regular representations of the demonstrated movement which are easier to interpret.

DR has many interesting features such as the possibility to visualize a learned model and a rigorous mathematical foundation. However, prior research on DR and imitation often focused on specific problem domains, such as locomotion. It did not provide a general approach suitable for learning different kinds of behaviors. Most importantly, the implementations already mentioned only allow for limited generalization and adaptation capabilities. In contrast, we will show in this thesis, how more complex adaptation capabilities can be achieved through the combination of dimensionality reduction with other methods.

2.3. Computer Animation

In computer animation, imitation is not in the focus of the scientific discussion. Yet, many problems that are tackled within this community are closely related to imitation. In particular, the synthesis, variation and adaptation of motion sequences is of great importance to computer animation. Synthesized motion sequences are often based on example data recorded by motion capture. The process of adapting observed data to new situations, characters, environments is reminiscent of imitation in humans and animals.

Modeling stylistic variations An interesting line of research within the computer animation community is concerned with algorithms for modeling stylistic features of human motion. The goal of this research is to increase the realism of virtual humans by generating variations and distinct styles of movements. For instance, a simple walking gait can be turned into a sneaky, or a joyful walk. Most important, this research tries to avoid repetitiveness in a virtual character's motions. In current computer games, for example, only a relatively small number of different motion clips is used for animation. However, the *exact* repetition of these clips at different times, renders the generated movements unnatural or 'robotic'. An early approach to the synthesis of varying human motion is presented in [PG96]. Perlin uses an additive noise component on top of procedurally generated animations, in order to make them appear more realistic. He

introduces a special type of noise function to this end. However, in order to make the noise match the underlying animation, some parameter tuning is necessary.

In [PB02], style is added to a keyframed animation by decomposing it into fragments, such that each fragment can be replaced by a similar motion from a motion capture library. The user can select important joints, which should drive the replacement process. A matching algorithm then finds motion clips which are similar with respect to the selected joints. After the matching, the motion capture data is merged, in order to remove any discontinuities. An interesting property of this approach is that it requires only a subset of joints to be specified. For example, when animating a walking gait, it is sufficient to specify the joint angles of the legs only. The remaining degrees of freedom and stylistic features are automatically added. A different approach is described in [LWS02]. Here, a statistical model of the animation is learned using linear dynamical systems, from which later new, slightly different motions can be synthesized. The main advantage of such data-driven approaches is that they reduce the amount of parameter tuning needed to generate natural-looking animation significantly. The important information is extracted from the given motion capture data. This ensures that synthesized variations obey the same dynamics as the original data. Recently, a similar approach using bayesian networks was presented [LBJK09]. In contrast to other methods, bayesian networks can synthesize new variants from a single example motion.

In [BH00] a powerful approach for synthesizing animation in a broad variety of styles was presented. The so-called *style machines* are generative probabilistic models that interpolate and extrapolate stylistic variations learned from example data. Basically, style machines are a special type of hidden Markov models. In contrast to traditional hidden Markov model however, style machines include an additional parameter. During synthesis, this parameter allows the user to modulate the style of the generated animation. In contrast to some of the approaches discussed earlier, style machines do not require the training data to be segmented into fragments. Other work targeting style extraction and translation was presented by Shapiro et al. [SCF06] and Hsu et al. [HPP05]. Style-based inverse kinematics [GMHP04] showed that data-driven approaches can help us to find new solutions to well studied problems. Originally, the problem of inverse kinematics (IK) is tackled by using geometric or iterative methods. However, the results achieved with these techniques often appear unnatural or anatomically infeasible. Furthermore, these approaches are typically restricted to individual limbs. Style-based IK, however, generates realistic, full-body solutions. A low-dimensional model is derived from a set of example postures. The model is the used to search for postures satisfying the IK constraints.

Instead of focusing on synthesis, the work in [RPE+05] targets the analysis of given animations by defining methods for quantifying the naturalness of motion. Using these methods it is possible to estimate whether an animation appears natural or not. For example, this can be used to verify if a motion editing step preserves the naturalness of a recorded motion. The approach uses machine learning techniques to extract a likelihood function from a large corpus of motion capture data. This likelihood function returns

the probability that an input motion sequence could be generated by the same statistical model as the training data. It assumes that motions that occur repeatedly are judged natural, whereas motions that happen rarely are not.

Our work is closely related to above approaches. We seek to develop efficient techniques that allow us to generate variations of recorded motion. We are also interested in quantifying the naturalness of the results. However, the *goal* of our work differs fundamentally from above efforts: our primary goal is not style translation or modification, but rather adaptation. We want to be able to produce variants of a demonstrated behavior, from which we can pick the ones which best fit the demands imposed by a new context.

Adaptation and Retargeting Retargeting means the adaptation of an existing animated motion from one setting to another or one character to another. The retargeting problem has attracted a substantial amount of research. For example, in [Gle98] a method is introduced which uses optimization techniques to adapt a given motion to a different character while retaining its distinct look. Additional constraints, so called spacetime constraints, are formulated and solved using nonlinear optimization techniques. Spacetime constraints specify the desired positions or orientations of an end-effector. Solving the optimization problem results in a deformation of the original motion, which satisfies the posed constraints. Using this approach, it is possible to force a character to walk in predefined footsteps, even if the lengths of its legs are changed. In [LS99], a hierarchical solution to this problem was introduced, which avoids solving large optimization problems by using inverse kinematics on a local scale. Recently, Hecker et al. [HRE+08] proposed a real-time motion retargeting technique which handles arbitrary morphologies of virtual characters. This approach is particularly interesting for application domains where much of the content is user generated, e.g. video games. Inverse kinematics approaches have also been successfully used in [KSG02; IAF06] to solve the problem of *footskating*. Footskating refers to situations where a character's foot unnaturally penetrates or slides across the ground. The character's leg and foot configuration are adjusted to ensure that ground contact is maintained.

A different approach is introduced by motion warping [WP95]. Motion warping models the angles of each joint as a cardinal spline encoding the angular value as a function of time. The parameters of these splines can be modified to create variations of the original motion. During a motion warping session, a user selects a frame in the animation and poses the model interactively. Then the splines of the original motion are deformed such that their curves pass through the joint values derived from the desired pose. Motion warping can therefore be seen as a simple blending or interpolation operation. A similar approach is introduced in [BW95]. Here, an animation is regarded as a set of one-dimensional curves on which signal processing methods can be applied. Bruderlin and Williams show that operations such as multiresolution filtering, waveshaping or the addition of smooth displacement maps can be used to modify a given animation. In a similar manner, Unuma et al. [UAT95] applied Fourier-analysis to this task.

Many motion blending techniques synthesize new animations by reassembling and interpolating between pieces from a motion database. For example, the verbs and adverbs approach of Rose et al. [RCB98] allows a user to interactively generate new animations based on interpolation of existing motion clips. Research on blending techniques also lead to the development of *motion graphs* [KGP02]. A motion graph is a directed graph used to represent the sequence of actions during an animation. The edges of a motion graph are motion fragments, whereas nodes correspond to choice points that connect different clips. Motion graphs are used to rearrange and seamlessly blend short motion fragments in order to synthesize a large number of new movements.

Most of the above motion techniques focus on simple kinematic constraints, such as ensuring particular end-effector positions. These approaches are not suitable for complex retargeting tasks, such as retargeting a recorded grasp to new objects. In the case of grasping, the relationship between the fingers and the object and the interdependency of the fingers has to be reflected in the optimization. Point and angular constraints are not sufficient to model a realistic and stable grasp. Additionally, it must always be ensured that the resulting grasps are feasible within the human anatomy. In contrast to these methods, this thesis seeks to develop techniques that are not limited to simple retargeting problems. We are interested in retargeting grasps to new objects, adapting walking gaits to new terrain, and the interactive adaptation to a human interaction partner.

Physics and Optimization Physics-based approaches to computer animation [WB97] simulate Newtonian mechanics in order to create dynamic environments. In addition to geometric properties, they also allow for the specification of physical properties of an object, including its mass or elasticity. Animations are then generated by simulating physical phenomena acting in the virtual environment, such as forces, torques or energies. This approach spares the animator of having to specify all required animations beforehands. Additionally, it allows for realistic, interactive environments. The backside of this, however, is that the algorithmic control of simulated characters becomes highly complex. More specifically, characters in physically simulated environments need to be actuated in terms of forces and torques, rather than in terms of simple joint angle differences.

One way to overcome this problem is to implement special-purpose controllers for each of the behaviors in the character's repertoire. A controller takes the current state of the character (and possibly also the state of the environment) and computes the torques necessary to accomplish the intended task. Controllers have long been used in robotics to drive autonomous robots. In the last 15 years a number of controllers have been developed in the computer animation literature, including controllers for standing up after falling [FvT01], for balance recovery [SCCH09], walking [YLv07], or cycling [HWBO95]. Most of these controllers are hand-crafted and based on biomechanical principles. While this procedure leads to physically plausible and highly realistic animations, often expert knowledge is needed to tune the parameters of these controllers. A less labour inten-

sive approach to the synthesis of physics-based animation was first proposed in [WK88]. Witkin and Kass called their approach *spacetime optimization*. It uses numerical optimization techniques in order to automatically derive the forces needed to actuate a virtual character. Important aspects and goals of the animation are specified in terms of constraints that are later satisfied by the numerical optimization process. For example, the animator can specify constraints for the location of a virtual character at the start and end of an animation. A constraint-based optimization technique then searches for an animation that satisfies the constraints and minimizes a given objective function simultaneously. Objective functions penalize non-physical and ineffective movements. Objective functions are typically based on some physical criterion, e.g. minimal torque change [UKS89], muscle exertion [RGBC96], or joint angle acceleration [FP03]. Over the years, the original spacetime approach was refined by many other researchers. For instance, in [LHP05] muscle preferences and joint stiffness are extracted from experimental data and used to parametrize the physical environment.

Spacetime optimization is generally a promising approach to the motion synthesis problem. The animator can concentrate on the artistic side of the animation, while a numerical procedure ensures physical plausibility and realism. This approach can synthesize animations which would normally require highly skilled animators. In their original paper, Witkin and Kass show that the synthesized motions obey many of the principles of animation[1] as used in the film industry. Still, spacetime optimization remains to date 'the realm of academic research' [KA08]. Partly, this is due to the fact that simulation of physical environments is a highly complicated process. The most important reason, however, is the underlying nonlinear optimization. Optimization is typically a time-consuming process. Especially for complicated humanoid characters, optimization can take a considerable amount of time for convergence. Furthermore, convergence towards an acceptable solution is not guaranteed. Therefore, in recent years, research is focused on accelerating this process. For example, in [SHP04] the computational complexity of optimization can be reduced by biasing the search process towards natural human motion. In addition to efficiency problems, another difficulty when using spacetime optimization are the required optimization constraints. The constraints have to be specified by the user in terms of key-frames. For example, to synthesize a walking motion, a set of postures must be specified along with information about timing and location in the animation. The optimizer generates a physically plausible motion that smoothly transitions between the key-frame constraints. This procedure considerably reduces the complexity of the animation process, in particular in contrast to traditional animation techniques, but it still requires the detailed specification of constraints by a human user. We will show in this thesis, that imitation learning can help to automatically extract important constraints from training data. Optimization can then be performed on a more abstract

[1] The twelve principles of animation were formulated by Johnston and Thomas in [TJ81]. While working at the Disney studios they identified a set of rules or principles that can guide the animation process. These rules have been adopted by other studios and practitioners and are considered, to this day, the necessary ingredients for high quality animation.

level. For example, when synthesizing fast walking motions for a robotic agent, we only need to specify that a goal position needs to be reached as fast as possible.

2.4. Conclusion

The Computer animation and robotics share a common ground—the control of synthetic humanoids such that they exhibit human-like behavior. Over time, each field developed domain-specific approaches to this problem. At the same time, the way humans acquire motor skills through imitation can be a good source of inspiration for both fields. Our work draws upon concepts from natural imitation to tackle the problem of motor skill acquisition and synthesis. Methods from both domains will be combined within a general approach.

Our new approach aims to overcome several limitations of the techniques discussed in this chapter. First of all, most of the above techniques are tailored for a specific type of synthetic humanoid, e.g. animated characters or robots. The transfer to other domains and other types of synthetic humanoids is non-trivial in most cases. We are interested in a general approach that can deal with arbitrary realizations of synthetic humanoids, including both simulated and physical agents. Another shortcoming of existing techniques are the limited capabilities for the adaptation to new situations. Retargeting approaches used in computer animation are often restricted to simple kinematic constraints and cannot deal with adaptation to complex objects or environments. In this regard it is also important to ensure that adaptations produce natural-looking, realistic postures and motions. Most retargeting techniques do not explicitly adress this problem. Efficiency is another important aspect that limits the application of several techniques discussed, for instance in the case of spacetime optimization. Especially for application in the computer animation domain, techniques with low computational demands are important in order to ensure interactivity. Therefore, any new algorithm for motion synthesis should feature a low response time.

3. Mathematical Foundations

In this section we will review the mathematical concepts which are fundamental for the approach presented in this thesis and for the way humanoid structures are modeled and animated. We will begin with a discussion of kinematic chains and rotation representations and then move on to discuss dimensionality reduction techniques in detail.

3.1. Articulated Structures

The human body is a highly complex structure that is articulated by an underlying skeletal and muscular system. Through the interplay of these systems, humans are capable of performing a wide range of movements and postures. Complex articulated objects, such as virtual humans or humanoid robots, are modeled by means of kinematic chains in robotics and computer animation.

3.1.1. Kinematic Chains

A kinematic chain is a hierarchical system of *segments* which are connected by *joints*. Segments are rigid bodies that represent body parts of a synthetic humanoid. In contrast, joints determine the configuration and, hence, the motion of the kinematic chain. Each joint connects two segments and constrains their relative motion. There are various types of joints, which mainly differ in the number of rotational degrees of freedom (DOF). For example, ball joints produce rotations about any axis in a three-dimensional space. In contrast, hinge joints only produce rotations about one axis.

The set of alternating joints and limbs forms a hierarchical, tree-like structure. Starting from a root node, segments and joints are attached as child nodes, which in turn can have futher child nodes. Changing the configuration of a parent node affects the underlying subtree. For example, by moving the shoulder joint, the position and orientation of upper arm, lower arm, and hand are also changed. The configuration of child nodes is specified relative to the configuration of the parent node. This is achieved using the concept of local coordinate systems.

Figure 3.1 illustrates the concept for a simple humanoid shape. The *end effector* represents one end of the kinematic chain, e.g. a hand. The position of this end effector is specified in terms of the local coordinate system centered at the elbow joint. Similarly, the position and orientation of the elbow joint is specified locally with respect to the coordinate system of the shoulder, which in turn is specified locally with respect to the

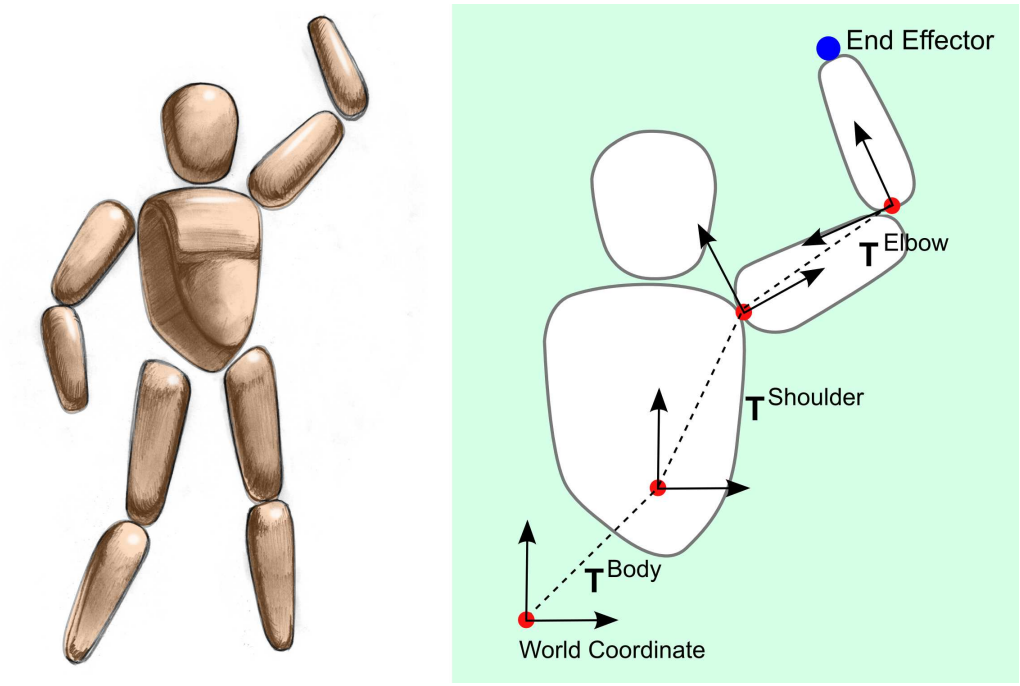


Figure 3.1: A stack of local coordinate systems realizes a kinematic chain for a simple humanoid wood figure.

coordinate system of the figure's body. If we move the shoulder of the humanoid figure, we want the position of the end effector should move accordingly. This is implemented by using a stack of geometric transformations.

The configuration of each joint is represented by a corresponding transformation. In a three-dimensional space a transformation can be written as a 4×4 matrix, having a rotational and a translational component. In Figure 3.1, the translational component between the coordinate systems is depicted using dashed lines. As explained before, the transformations are relative to each other. That is, to determine the configuration of a joint in the kinematic chain, all previous transformations need to be accumulated. For example, given the position $\boldsymbol{r}$ of the end effector in coordinates of the elbow joint, the following equation is needed to compute its global position:

$$\boldsymbol{r}^{\text{Global}} = \mathsf{T}^{\text{Body}} \cdot \mathsf{T}^{\text{Shoulder}} \cdot \mathsf{T}^{\text{Elbow}} \cdot \boldsymbol{r} \tag{3.1}$$

Note, that in this case the torso is the root of the kinematic chain, and that the transformation T^{Body} is needed to position and orient the whole figure with respect to the global coordinate system. The use of local coordinate systems ensures that limbs stay attached to each other and simplifies the animation process significantly.

3.1.2. Rotation Representation

The transformations involved in the computation of kinematic chains contain a translational and a rotational component. The translation is always specified in terms of a three-dimensional vector. In contrast, there exists a variety of different representations that can be used for specifying rotations in space. Each of these representations has it advantages and drawbacks depending on the desired application.

The fundamental theorem on rotation by Leonhard Euler (1707-1783) states that:

> "Any rigid motion leaving a point fixed may be represented by a singular planar rotation about a suitable axis passing through that point" [BT03]

As a consequence of this theorem, any rotation in space can be specified using an axis of rotation $\boldsymbol{v}$ and a rotation angle φ. The pair $(\boldsymbol{v}, \varphi)$ is called the *axis-angle* representation of the corresponding rotation.

Another type of rotation representation, also introduced by Euler, is based on his proof that any 3D rotation can be expressed as a sequence of three rotations about the coordinate axes. These angles are known as Euler angles. There are various conventions in the literature on the axes about which to rotate and on the order in which rotations must take place. In the following we will consider the *heading-attitude-bank* convention, resulting in the following rotation matrix R^E:

$$\mathsf{R}^E = \left(\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \tag{3.2}$$

The representation by Euler angles exhibits a singularity of orientation at $\varphi = \pm\frac{\pi}{2}$, resulting in the loss of one rotational DOF. This phenomenon is often termed *gimbal lock*. The gimbal lock can be avoided by using the quaternion representation. Quaternions are a noncommutative extension of complex numbers, which are particularly well suited for the representation and interpolation of rotations. A quaternion is a vector of four elements $(w, x, y, z)^T$ and can easily be derived from the axis-angle notation using the following equation:

$$(w, x, y, z)^T = (\cos\frac{\varphi}{2}, \boldsymbol{v}_x \sin\frac{\varphi}{2}, \boldsymbol{v}_y \sin\frac{\varphi}{2}, \boldsymbol{v}_z \sin\frac{\varphi}{2})^T \tag{3.3}$$

Quaternions have become a de facto standard in modeling and animation of humanoid characters in computer graphics. However, in recent years various researchers suggested the use of an *exponential map* representation for rotations.

An exponential map specifies a rotation by the parameters a, b, and c with:

$$\varphi^2 = a^2 + b^2 + c^2 \tag{3.4}$$

$$\boldsymbol{v} = \frac{1}{\varphi}(a, b, c)^T \tag{3.5}$$

Each of the above five rotation representations has its strength and weaknesses. Matrices, for instance, are difficult to interpolate and invert, while Euler angles suffer of the above mentioned gimbal lock. According to Grassia [Gra98]: "no single parametrization is best for all applications". Hence, the choice of rotation representation should be carefully made based on the target application.

3.2. Dimensionality Reduction

The goal of dimensionality reduction (DR) is to extract hidden structure from a given data set. The underlying assumption is, that not all measured variables are important and that many of them are correlated. By analyzing the intrinsic variability of a data set, we can understand these correlations and, accordingly, remove all redundancies. For instance, if we imagine the surface of the earth as a hollow sphere, then the coordinates of each point on its surface can be specified by a three-dimensional vector in a Cartesian coordinate system. At the same time, we can also specify the exact coordinates of each point with only two dimensions, if we use the latitude-longitude representation. This becomes obvious if we remember that maps of the earth are printed on sheets of paper. Such a map can be seen as a projection of the original 3D coordinates onto a 2D space. Hence, the three-dimensional, Cartesian specification of positions on the surface of the earth is redundant. DR methods can be used whenever similar redundancies in a data set need to be automatically extracted to derive a compressed representation.

DR also allows us to perform exploratory data analysis on any given high-dimensional data set. For inspection and analysis purposes it is convenient to create a visual representation of the data at hand. The resulting visualization helps to *explore* the data and gain valuable insights into its structure. However, data sets with more than three or four dimensions are difficult to visualize. We can overcome this problem by first projecting the data to a $2D$ or $3D$ space to visualize the result. This can help us to better understand the type of data gathered. In turn, an informed choice on the ideal algorithm for processing or classifying the data can be made.

In the following we will introduce basic mathematical concepts and notations which will be used throughout this thesis. We will then present various DR algorithms including well known methods and state-of-the-art techniques. Finally, we will compare the methods presented and discuss their properties.

3.2.1. Basic Concepts

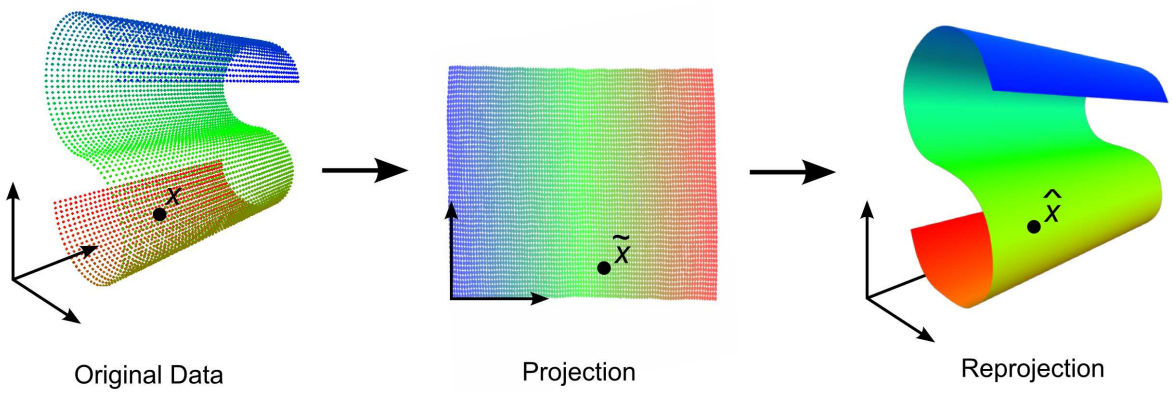


Figure 3.2: Dimensionality reduction applied to a set of data points, sampled from a S-shaped surface. The process projects the initially three-dimensional points into a two-dimensional space. Projecting a point $\boldsymbol{x}$ on the manifold from the 3D space to the 2D image space yields image coordinates $\widetilde{\boldsymbol{x}}$. Reprojecting $\widetilde{\boldsymbol{x}}$ yields the point $\widehat{\boldsymbol{x}}$ in 3D space.

In Figure 3.2 we see an example of a dimensionality reduction process. On the left a set of data points that are distributed on a smooth surface with a shape of the letter S are shown. Although the points exist within a three-dimensional vector space, by definition they reside on a two-dimensional surface. For a better understanding of the latter statement, we can imagine the points as lying on a slightly folded sheet of paper. Therefore, they have only two modes of variability, i.e. their coordinates can be expressed in terms of a two-dimensional vector. The data points lie on a two-dimensional *manifold* embedded in a space of higher dimensionality. The concept of manifolds is an important concept when working with DR techniques. Formally, a manifold is defined as [Fom09]:

Definition 1 (Manifold). *A set $\mathcal{M}$ is called a L-dimensional* manifold *if it is locally homeomorphic with $\mathbb{R}^L$. That is, if for every $x \in \mathcal{X}$ there exists a neighborhood $\mathcal{Y}_x \in \mathcal{M}$ and a homeomorphism $f : \mathcal{Y}_x \to \mathbb{R}^L$.*

where homeomorphism is defined as follows:

Definition 2 (Homeomorphism). *A function $f : \mathcal{X} \to \mathcal{Y}$ is a homeomorphism if f is continuous and invertible and the inverse function f^{-1} is also continuous. The spaces $\mathcal{X}$ and $\mathcal{Y}$ are then topologically equivalent.*

The above definition can be translated into the following: a manifold can be divided into a set of patches, with each patch having dimensionality L. The patch is said to be locally L-dimensional. For example, a hollow sphere is locally a surface, hence, a two-dimensional manifold, yet, it exists in a three-dimensional space. In summary, the sphere

is a two-dimensional manifold embedded in a three-dimensional space. Dimensionality reduction techniques are used to automatically extract manifolds from a given set of input data and represent them with as few variables as possible. The result is a condensed representation of the manifold. If we apply this idea to the introduced S-curve, this means that (ideally) by using DR techniques, we reduce our data to a $2D$ space, in which all points lie on a flat surface, see 3.2(b).

Definition 3 (Dimensionality Reduction)**.** *Let* $\mathcal{X} = \{\boldsymbol{x}(0), .., \boldsymbol{x}(N) |\ \boldsymbol{x}(i) \in \mathbb{R}^H\}$ *be a set of data points sampled from a* L*-dimensional manifold embedded in a* H*-dimensional space, where* $L \ll H$ *holds. Dimensionality reduction projects* $\mathcal{X}$ *to a low-dimensional vector space with dimensionality* L*, leading to a new set of points* $\widetilde{\mathcal{X}} = \{\widetilde{\boldsymbol{x}}(0), .., \widetilde{\boldsymbol{x}}(N) |\ \widetilde{\boldsymbol{x}}(i) \in \mathbb{R}^L\}$*. A point* $\widetilde{\boldsymbol{x}}(i)$ *is called the lower-dimensional* image *of* $\boldsymbol{x}(i)$*.*

Depending on the technique applied, the projection can have different objectives or goals. Many techniques try to minimize the so-called *reprojection-error*. When projecting a set of points to a low-dimensional manifold and back to the original space, the original data can ideally be perfectly reconstructed. While in general this is not realistic, one can still try to minimize the loss of information introduced by this process. The difference between the set of points $\mathcal{X}$ and the reprojected counterpart $\widehat{\mathcal{X}}$ should be as small as possible with respect to a specified distance metric. The tilde and the hat are henceforth used for specifying low-dimensional and high-dimensional points, respectively. We will therefore adopt the following notation:

- Projected data will be marked by a tilde, e.g. $\widetilde{\boldsymbol{x}}$
- Reprojected data will be marked by a hat, e.g. $\widehat{\boldsymbol{x}}$

An important feature of dimensionality reduction techniques is the fact that *continuous*, smooth manifolds are derived from a discrete set of input data. As a result, we can use the learned manifold to synthesize new points that contain the same type of correlations between the variables as the input data. For the case of the S-curve example this means, that we can parametrize an arbitrary point on the S-curve by specifying its two-dimensional image coordinates and then reprojecting it back into the original three-dimensional space. The synthesized points can be regarded as interpolations and extrapolations of the input data. Figure 3.2 shows an example of this process: the image point $\widetilde{\boldsymbol{x}}$ is reprojected, leading to a point $\widehat{\boldsymbol{x}}$ on the surface of the S-curve. This feature of dimensionality reduction techniques is particularly important, as it allows us to create simple synthesis models for any given data set.

3.2.2. Dimensionality Reduction Methods

In this section we will introduce seven important algorithms for DR. This includes well-known algorithms such as PCA and MDS, as well as current state-of-the-art techniques including Isomap and LLE.

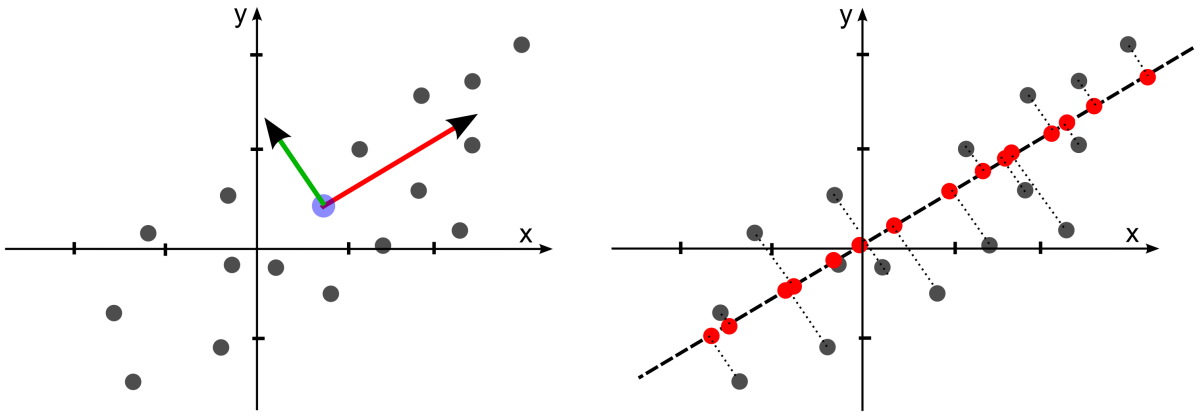


Figure 3.3: Principal Component Analysis determines the directions of highest variability in the set of data points. Left: The red and green arrows show the first and second principal component, respectively. Right: The data points can be projected onto the first principal component in order to reduce dimensionality while retaining as much of the inherent information as possible.

Principal Component Analysis

Principal component analysis (PCA), also known as Hotelling transform or Karhunen-Leuve transform, tries to reduce the dimensionality of the data while retaining as much as possible of the contained information. This is done by searching for a linear subspace with a lower number of dimensions L along which the data has maximum variance. PCA extracts L orthogonal vectors, the so-called principal components, which form the basis (the coordinate system) of the linear subspace. The data points can be reconstructed using a linear combination of these principal components.

Figure 3.3 shows the application of PCA on data points sampled from a Gaussian distribution. As already mentioned, PCA extracts the principal axes with the goal of retaining as much of the variance as possible. Therefore, the first principal component is in the direction where the data cloud exhibits the highest variability (red arrow). The second principal component is in direction with second highest variability (green arrow), and so on. The amount of spread in each direction can be used to assess the relative importance of the corresponding principal component. This can be used to discard principal components that have a low contribution.

To compute a PCA, we need to derive the covariance matrix Σ of the data, which requires the estimation of the sample mean $\boldsymbol{\mu}$ of the data cloud. This is achieved by:

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}(i) \tag{3.6}$$

$$\Sigma_{ij} = \frac{1}{N-1}\sum_{k=1}^{N}(\mathsf{X}_{ki}-\mu_i)(\mathsf{X}_{kj}-\mu_j) \tag{3.7}$$

where X is a matrix of the vectors $\boldsymbol{x}(i)$ as rows. The covariance matrix Σ contains information on the variability of the data in different dimensions. The matrix is factored using *spectral decomposition*, also called eigenvalue decomposition (EVD). This yields matrices V and Λ so that :

$$\Sigma = \mathsf{V}\Lambda\mathsf{V}^T \tag{3.8}$$

where V is a square matrix whose columns are unit norm *eigenvectors* of Σ, and Λ is a diagonal matrix containing the eigenvalues λ_i of Σ. The computed eigenvectors hold the principal components of the data set, while the eigenvalues store the variance of the data in each of the components. The eigenvalues can be used to determine the contribution of a given principal component to the overall information contained in the data. For instance, the first principal component—the principal component which captures the largest part of the information—is the eigenvector with the largest associated eigenvalue. The fraction:

$$\frac{\sum_{i=1}^{L}\lambda_i}{\sum_{i=1}^{H}\lambda_i} \tag{3.9}$$

gives the cumulative proportion of the variance, which is the amount of information captured by the first L principal components. For example, if we want to retain 95% of the contained information, we need to keep the first L principal components for which the calculation of the above fraction yields a value higher than 0.95. By analyzing this fraction for different values of L, we estimate the intrinsic dimensionality of the dataset. If the data is correlated, most of the information can be retained using only few principal components ($L \ll H$).

The computed principal components form the basis of the lower-dimensional subspace on which to project the data points. We can project any point from the original space into this subspace. Conversely, any point from the subspace can be projected back into the original space. The two operations, henceforth called projection and re-projection, are arguably the most important operations when using dimensionality reduction techniques. The projection from the original space to the subspace can be achieved by first substracting the sample mean $\boldsymbol{\mu}$ and then computing the dot product of the result with each of the L principal components. Each dot product returns the coordinate with respect to the corresponding principal component. For example, in Figure 3.3 we see how we can project each of the 2D points onto a 1D subspace. This is achieved by removing

the mean value and then calculating the dot product with the first principal component. In matrix notation the operation can be expressed as:

$$\widetilde{\boldsymbol{x}} = \mathsf{W}^T(\boldsymbol{x} - \boldsymbol{\mu}) \tag{3.10}$$

where W is a matrix containing only the first L eigenvectors as columns. If we want to reproject a point from the subspace back into the original space, we need to multiply each coordinate value with the corresponding principal component. The sum of these vectors plus the sample mean will be the searched vector in the high-dimensional space. This can be expressed as:

$$\widehat{\boldsymbol{x}} = \boldsymbol{\mu} + \mathsf{W}\boldsymbol{y} \tag{3.11}$$

PCA has several features that make it applicable in many situations. The mathematics behind the technique is easy to understand and to implement. Furthermore, this technique does not require any kind of hyperparameters found in other DR methods.

Multidimensional Scaling

Multidimensional scaling (MDS) is a classical technique in exploratory data analysis. While there exist many variations of the algorithm, in general the term is used to refer to the version due to Kruskal et al. [KW78]. MDS projects a data set to a lower-dimensional space with the focus on preserving the pairwise distances of the data points. Given two points $\boldsymbol{x}(i)$ and $\boldsymbol{x}(j)$, they are projected in such a way that their distance $\mathcal{D}(\boldsymbol{x}(i), \boldsymbol{x}(j))$ closely matches the distance $\mathcal{D}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j))$ of their image points $\widetilde{\boldsymbol{x}}(i)$ and $\widetilde{\boldsymbol{x}}(j)$. The distance preservation property allows to draw important conclusions about similarity or dissimilarity of data in the high-dimensional space by analyzing the projected counterpart.

MDS can be formulated as an optimization problem. The goal of optimization is to minimize a cost function called *stress*, which measures how well the distances in the high- and low-dimensional space fit. The stress function E is defined as:

$$E_{MDS} = \sqrt{\sum_{ij} \left[\mathcal{D}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j)) - \mathcal{D}(\boldsymbol{x}(i), \boldsymbol{x}(j))\right]^2} \tag{3.12}$$

A good projection is determined by a set of image points $\widetilde{\mathcal{X}}$, such that E_{MDS} is minimized. This can be achieved by randomly initializing $\widetilde{\mathcal{X}}$ and then using gradient descent on the E_{MDS} function. In each step of the algorithm, the $\widetilde{\mathcal{X}}$ is changed in the direction of the highest rate of decrease in the the stress function. After several iterations, the algorithm converges towards the optimal image points. Another way of computing an MDS works as follows. First, we compute the matrix of squared distances with elements D_{ij} according to equation:

$$\mathsf{D}_{ij} = \left[\sum_{k=1}^{H} (\mathsf{X}_{ik} - \mathsf{X}_{jk})^2\right] \quad \forall i, j \in \{1, .., N\} \tag{3.13}$$

where X is a matrix with $\boldsymbol{x}(i)$ in its rows. Note, that in this case the Euclidean distance metric is used for the computation of the D_{ij} values. Other metrics such as the Manhattan metric can also be used. Next, the entries of the matrix are centered according to formula:

$$\mathsf{B}_{ij} = -\frac{1}{2}\left[\mathsf{D}_{ij} - \frac{1}{N}\sum_{k=1}^{N}\mathsf{D}_{ik} - \sum_{k=1}^{N}\mathsf{D}_{kj} + \frac{1}{N^2}\sum_{k=1}^{N}\sum_{l=1}^{N}\mathsf{D}_{kl}\right] \quad \forall i, j \in \{1, .., N\} \tag{3.14}$$

yielding matrix B of centered distances. The eigenvectors and eigenvalues of matrix B can be calculated using EVD, as already discussed for PCA. The projection, or more specifically the matrix of image points, can then be computed according to equation:

$$\widetilde{\mathsf{X}}_{ij} = \sqrt{\boldsymbol{\lambda}_j}\mathsf{V}_{ij} \quad \forall i \in \{1, .., N\}, j \in \{1, .., L\} \tag{3.15}$$

where matrix V is derived from the eigenvalue decomposition. An interesting point about the algorithm, is that the projection does not necessarily require the coordinates of the points to be projected. The interpoint distances are sufficient for computing the embedding. Due to this feature, MDS is particularly attractive for social sciences. People are asked to rate the similarity between objects or other entities of the study. The acquired similarities can then be processed using MDS to create a low-dimensional visual representation showing the distances between these entities.

Locally Linear Embedding

Although the earth has a sphere-like geometry, for a long time it was believed to be a flat surface. The reason for this misconception is the fact that smooth, nonlinear manifolds appear linear on small scales. This idea is the basis for the locally linear embedding (LLE) algorithm [RS00]. LLE tries to recover the global structure of a manifold using a collection of locally linear patches. The goal is to identify these patches and find a mapping to a space of lower dimensionality that preserves the neighborhood of points.

The first task in computing an LLE consists of characterizing the local geometry of the patches by linear coefficients that reconstruct each data point from a set of nearest neighbors. The number K of nearest neighbors is a parameter of the algorithm, which has to be provided by the user. If the data set is sufficiently large, we can assume that a value for K exists, such that the data point and its neighbors lie on or are close to a locally linear patch of the manifold. The local geometry of these patches is then characterized by linear coefficients that reconstruct each data point from its neighbors.

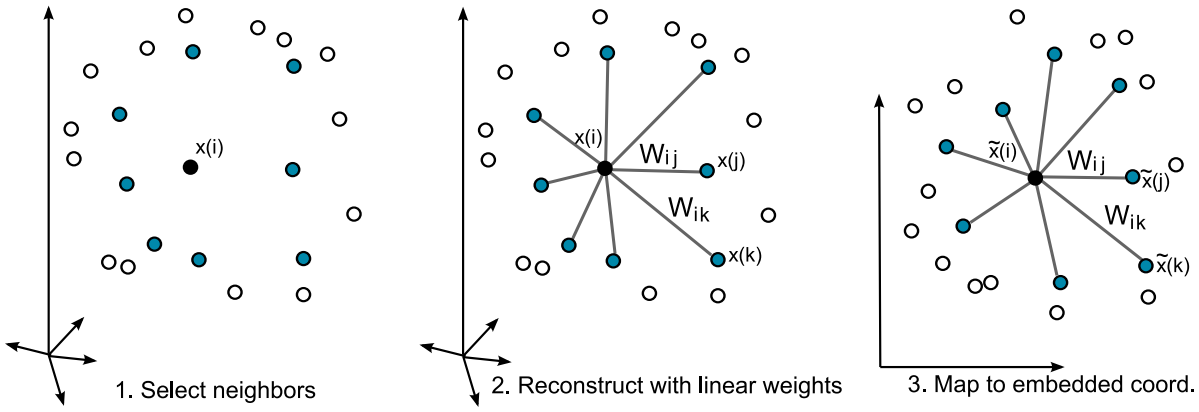


Figure 3.4: The steps involved in computing a locally linear embedding (based on [RS00]).

The reconstruction error that results from this process can be measured by:

$$E_{LLE} = \sum_{i}^{N} \left\| \boldsymbol{x}(i) - \sum_{j} \mathsf{W}_{ij}\ \boldsymbol{x}(j) \right\|^2 \tag{3.16}$$

where the entry W_{ij} of matrix W holds the contribution of the jth data point to the ith reconstruction. The cost function E_{LLE} computes the squared error between a point and its locally linear reconstruction. In order to compute the weights W_{ij}, the function E_{LLE} is minimized subject to the following two constraints:

- Each point $\boldsymbol{x}(i)$ is reconstructed solely from its neighbors, i.e. if a point $\boldsymbol{x}(j)$ does not belong to the set of neighbors, then the weights are zero.
- Each row of the weight matrix must sum to one: $\sum_{j=1}^{N} \mathsf{W}_{ij} = 1$. This means that each point is reconstructed from a convex combination of its neighbors.

These two constraints ensure an important property of the computed weights. For any data point $\boldsymbol{x}(i)$ the weights are invariant to rotation, scaling and translation. The weights characterize intrinsic geometric properties of each locally linear patch. These properties are valid in both the original high-dimensional space as well as the low-dimensional embedding of the data. The process of LLE can be understood as taking each locally linear patch and then placing it on a flat surface of lower dimensionality. In other words, a linear mapping consisting of a translation, rotation and rescaling exists, that maps the high-dimensional coordinates of each neighborhood to global internal coordinates on the manifold. By design, the reconstruction weights W_{ij} are invariant to linear transformations. The same weights that reconstruct a point in a high-dimensional space of dimensionality H can also reconstruct its manifold coordinates in a L-dimensional embedding space.

Algorithm 1 Projection by linear interpolation: Given an point to $\boldsymbol{x}$ to be projected, the matrix of all training data X, and the matrix of image coordinates $\widetilde{\mathsf{X}}$, the algorithm computes the projected point $\widetilde{\boldsymbol{x}}$.

Require: $(\boldsymbol{x} \in \mathbb{R}^H, \mathsf{X} \in \mathbb{R}^{H \times N}, \widetilde{\mathsf{X}} \in \mathbb{R}^{L \times N}, K)$

1: Select K neighbors $\mathsf{Y} \in \mathbb{R}^{H \times K}$ of $\boldsymbol{x}$ from set $\mathcal{X}$

2: Create the image point matrix $\widetilde{\mathsf{Y}} \in \mathbb{R}^{L \times K}$

3: Compute covariance matrix $\mathsf{\Sigma}$:

$$\mathsf{\Sigma}_{ij} = \frac{1}{N-1} \sum_{k=1}^{N} (\mathsf{Y}_{ik} - \boldsymbol{x}_k)(\mathsf{Y}_{jk} - \boldsymbol{x}_k) \quad \forall i, j \in \{1, .., K\} \tag{3.17}$$

4: Compute weights W by solving the following equation:

$$1 \;=\; \sum_{j=1}^{K} \mathsf{\Sigma}_{ij} \mathsf{W}_j \quad \forall j \in \{1, .., K\} \tag{3.18}$$

5: Normalize weights:

$$\mathsf{W}_i \;=\; \frac{\mathsf{W}_i}{\sum_{j=1}^{K} \mathsf{W}_j} \quad \forall i \in \{1, .., K\} \tag{3.19}$$

6: Compute projection $\widetilde{\boldsymbol{x}} \in \mathbb{R}^L$ by:

$$\widetilde{\boldsymbol{x}} \;=\; \sum_{j=1}^{N} \mathsf{W}_j \widetilde{\mathsf{Y}}_{ji} \tag{3.20}$$

Given the optimized reconstruction weights W, the next step of the algorithm is to compute the coordinates of the low-dimensional points $\widetilde{\boldsymbol{x}}(i)$ which reside on the manifold. This is achieved by finding L-dimensional coordinates such that the following cost function is minimized:

$$E_{LLE}(\widetilde{\boldsymbol{x}}) = \sum_{i}^{N} \left\| \widetilde{\boldsymbol{x}}(i) - \sum_{j} \mathsf{W}_{ij}\ \widetilde{\boldsymbol{x}}(j) \right\|^2 \tag{3.21}$$

Obviously this equation closely resembles Equation 3.16 with the difference that the equation is not minimized for the weights W_{ij}, but rather for the points $\widetilde{\boldsymbol{x}}(i)$. The weights have been computed in the earlier stage and are now used to find optimal embedding coordinates. Another difference is the dimensionality. In contrast to Equation 3.16, which uses the original points with dimension D, Equation 3.21 works on the image

points, which have dimensionality L. The cost function in Equation 3.21 can also be formulated as:

$$E_{LLE}(\widetilde{\mathsf{X}}) = \widetilde{\mathsf{X}}^T \mathsf{M} \widetilde{\mathsf{X}} \tag{3.22}$$

with $\mathsf{M} = (\mathsf{I} - \mathsf{W})^T(\mathsf{I} - \mathsf{W})$ and I being the identity matrix. Figure 3.4 summarizes the steps involved in computing an LLE. First, each data point is assigned a set of K neighbors, which are nearest to it. Next, the weights W_{ij}, which best reconstruct $\boldsymbol{x}(i)$ from its neighbors, are computed. Finally, the image point $\widetilde{\boldsymbol{x}}(i)$ is found which minimizes equation 3.21. In this final step, the weights are fixed and only the coordinates of the image points are changed. In Algorithm 1 we find the general procedure for projecting a new point, the so-called out-of-sample extension.

Isometric Feature Mapping

Many methods for dimensionality reduction use distance preservation as a criterion for determining low-dimensional projections of data points. The image points are computed in a way that their distances are as close as possible to the distances of the original data points, like for the MDS algorithm introduced in Section 3.2.2. The isometric feature mapping (Isomap) algorithm [TdL00] builds on this idea of distance preservation and extends it to nonlinear manifolds.

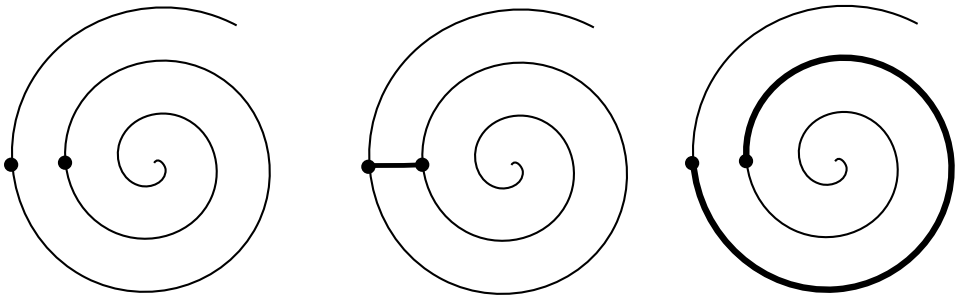


Figure 3.5: Dimensionality reduction builds on preserving interpoint distances, such as the distance of the two points on the spiral curve (left). The Euclidean distance between these points (middle) does not reflect their real distance with respect to the manifold. In contrast, the *geodesic* distance measure (right) takes the intrinsic geometry of the manifold into account (from [LLDV00]).

In the original MDS algorithm, a *Euclidean* distance metric is used to measure the pairwise distances of points. However, applying this metric to nonlinear manifolds leads to deceptive distance estimates. This fact is illustrated in Figure 3.5. The spiral is an example for a one-dimensional manifold embedded in a two-dimensional space. This becomes obvious if we image a spiral as a rolled straight line. Ideally, after application of a dimensionality reduction algorithm, the spiral would be unrolled to a straight line again. However, this unfolding is difficult for classical MDS because the pairwise Euclidean

distances after projection are larger than the Euclidean distances in the original high-dimensional space. The reason for this can be seen in Figure 3.5. For two arbitrary points the Euclidean metric in the high-dimensional space can go through shortcuts leading to a low distance value. This distance does not accurately reflect their intrinsic similarity. If we move along the manifold in order to get from one point to the other, we find the points to be much farther apart then indicated by the estimated distance. In order to accurately estimate the pairwise distance, we need to compute the shortest path between the points *along* the manifold. Such a distance metric is called a *geodesic* distance.

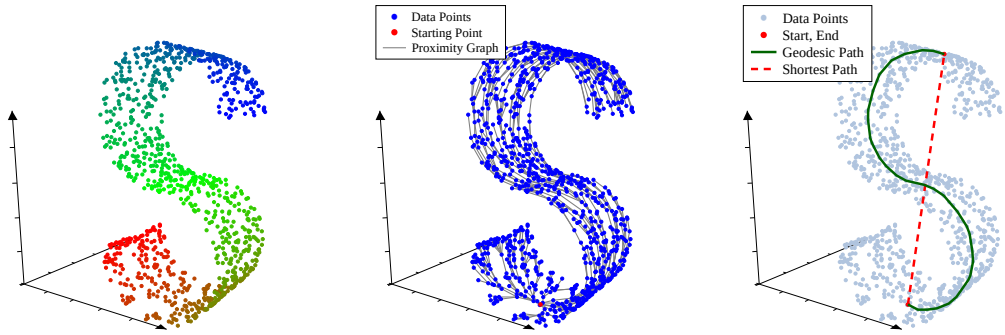


Figure 3.6: Left: Data points sampled from the S-curve benchmark. Middle: The proximity graph based of the data points. Right: The Euclidean- and the graph-distance for two given points.

The Isomap algorithm is based on this distance metric to transform classical MDS into a nonlinear dimensionality reduction technique. The difficulty, however, lies in estimating the geodesic distance. A priori we do not have any mathematical description of the manifold on which our data set resides. Therefore, we can not compute the true value of the geodesic distance of a given data pair. This problem is circumvented by approximating the geodesic distances using a so-called *graph distance* metric. The rationale behind the graph distance metric is twofold. First, for neighboring points, the Euclidean distance provides a good approximation to the geodesic distance. Second, for faraway points, the geodesic distance can be approximated by a sequence of short steps along neighboring data points. For this, a proximity graph is created as can be seen in Figure 3.6 (middle) for the S-curve benchmark data set. The nodes of this graph correspond to the points in our data set, while arcs connect neighboring points. Given the shortest path on this graph, we can estimate the geodesic distance by the sum of the arc length along the shortest path linking both points. The computation of the shortest path is typically computed using Dijkstra's [Dij59] algorithm. After approximating the geodesic distance between all pairs of points, classical MDS is applied to the resulting set of graph distances. This constructs an embedding of the data in a L-dimensional space that best preserves the manifold's intrinsic geometry. In other words, the distances of

the projected image points will reflect the geodesic distance of the original data points. The main difference between Isomap and MDS is the use of the graph distance metric. This difference, however, allows Isomap to unfold complex, nonlinear structures.

Sammon's Nonlinear Mapping

Sammon's nonlinear mapping (NLM) [Sam69], also referred to as Sammon mapping, is a widely used variant of the MDS algorithm (see Section 3.2.2). An important drawback of the classical MDS algorithm is that it focuses on retaining distances between faraway points, while small distances have little influence on the stress function. As a result, the low-dimensional embedding does not preserve the local structure of the manifold. However, in order to reveal the finegrained details of a given manifold it is often important to preserve such local structure. Thus, the NLM algorithm focuses on accurately mapping shorter distances rather than longer distances between data points. This is achieved by a weighting factor which is inversely proportional to the distance in the high-dimensional space. The stress function E_{NLM} which is minimized by NLM can be written as:

$$E_{NLM} = \frac{1}{\sum_{i=1}^{N}\sum_{j>i}^{N} \mathcal{D}\left(\boldsymbol{x}(i), \boldsymbol{x}(j)\right)} \sum_{i=1}^{N}\sum_{j>i}^{N} \frac{\left[\mathcal{D}\left(\boldsymbol{x}(i), \boldsymbol{x}(j)\right) - \mathcal{D}\left(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j)\right)\right]^2}{\mathcal{D}\left(\boldsymbol{x}(i), \boldsymbol{x}(j)\right)} \tag{3.23}$$

This stress function, however, can not be minimized in a closed form. For finding an optimal set of image points $\widetilde{\mathcal{X}}$ minimizing the above equation we have to use optimization techniques, such as gradient descent or Newton's method. First, the image points are initialized; either randomly or by first using classical MDS. Next, the gradient of the stress function is computed and the coordinates of the image points are updated in negative direction of the gradient. This procedure is iterated until the results of the stress function converge.

The main drawback of this approach is that these optimization algorithms do not guarantee convergence towards the global optimum and often get trapped in local optima. Additionally, this renders NLM sensitive to initial parameter settings. In other words, depending on the initial positions of the image points before optimization, NLM returns different results.

Curvilinear Component Analysis

Curvilinear component analysis (CCA) was proposed by Demartines and Herault [DH97] as an improvement of Kohonen's self-organizing map (SOM) algorithm [KSH01]. A SOM is an artificial neural network that performs vector quantization and dimensionality reduction at the same time. It consists of a discrete set of neurons arranged on a regular grid. Each neuron encodes a vector in the high-dimensional space of input data. The vectors are called prototype vectors. From a dimensionality reduction point of view the

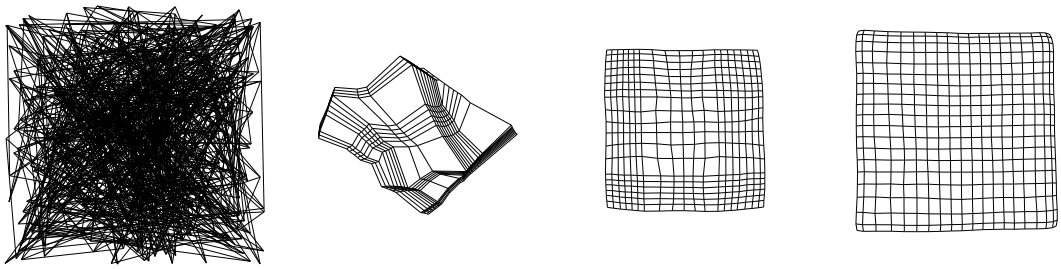


Figure 3.7: A Self-Organizing Map at iteration 0, 1000, 10000 and 40000 of learning (left to right). The process is reminiscent of a 'fishing net' slowly covering the data set.

prototype vector reflects the coordinate of the neuron in the high-dimensional space, while its position on the grid reflects the low-dimensional coordinate. During learning, the prototype vectors are modified to best fit the data cloud. In [LV07] this process is compared to covering the data using an elastic fishing net. This process is depicted in Figure 3.7. The projection of a SOM is both topology- and distribution-preserving. Topology preservation refers to the feature of mapping similar input points onto nearby neurons. Distribution preservation makes sure that more neurons are allocated to data points that appear more frequently in the input space.

A major drawback of SOM is that the lattice on which the data points are projected is fixed and must be specified by the user beforehands, i.e. the number of neurons, as well as the structure of the lattice, and a function describing neighborhood relationships between neurons has to be predefined. The output of a SOM is not a continuous low-dimensional space, but rather a discrete set of representative points along the manifold. Accordingly, it does not extract explicit mapping rules that allow us to project or reproject new points. Additionally, from a mathematical point of view the self-organizing map is still not well understood and convergence cannot be assessed. CCA adopts several ideas and concepts of SOMs, while at the same time addressing the shortcomings.

Similarly to SOM, CCA performs both vector quantization and dimensionality reduction. In the original work of Demartines et al. [DH97] both steps were performed simultaneously. In most modern implementations, however, these two steps are separated. Here, the first step of CCA consists of vector quantization. This can be performed using an arbitrary vector quantization algorithm such as Dynamic Vector Quantization. The vector quantization step drastically reduces the number of points which have to be used for dimensionality reduction and, thus, reduces computation time. At the same time the prototype vectors are computed in a way that they approximately retain the overall structure underlying the original data points. Once vector quantization is finished, the resulting prototype vectors are projected to the lower-dimensional embedding space.

This is achieved by using the following stress function:

$$E_{CCA} = \sum_{i=1}^{N} \sum_{j=1}^{N} \left[\mathcal{D}(\boldsymbol{x}(i), \boldsymbol{x}(j)) - \mathcal{D}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j)) \right]^2 \; \mathcal{H}_{\beta(t)}(\mathcal{D}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j))) \tag{3.24}$$

The function $\mathcal{H}_{\beta(t)}$ is used for weighting of the contribution of the distance of each pair of points. Generally, a bounded and monotonically decreasing function is chosen. The goal of introducing this factor is to favor the preservation of local topology, similarly to the neighborhood function used in SOM. A common choice for $\mathcal{H}_{\beta(t)}(\boldsymbol{x}(i), \boldsymbol{x}(j))$ is the Heaviside step function:

$$\mathcal{H}_{\beta(t)}(x) = \begin{cases} 1 & if \ \beta(t) \geqslant x \\ 0 & else \end{cases} \tag{3.25}$$

where $\beta(t)$ is a function of time. Typically, exponential or sigmoidal functions are used for this. Interestingly, the stress function of CCA closely resembles that of NLM. In fact both techniques share the same approach and methodology. In both cases DR is achieved through minimization of a given stress function. The main difference between these techniques is the weighting factor. NLM weights the projection error of each data pair based on their distance in the high-dimensional space. In contrast, CCA weights the projection error based on the distance in the low-dimensional projection space. While the result of $\mathcal{D}(\boldsymbol{x}(i), \boldsymbol{x}(j))$ does not change during the course of optimization, the result of $\mathcal{D}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j))$ changes with every optimization step. This idea is borrowed from the SOM algorithm and allows to unfold highly folded data sets. If the distances can be preserved, then it holds that $\mathcal{D}(\boldsymbol{x}(i), \boldsymbol{x}(j)) \approx \mathcal{D}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j))$. In this case CCA behaves similarly to NLM. However, if $\mathcal{D}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j)) \ll \mathcal{D}(\boldsymbol{x}(i), \boldsymbol{x}(j))$, then the manifold is highly folded and the image points must be drastically changed to unfold it. Therefore, the weighting function will assign a high contribution of the corresponding data pair to the overall stress function.

Curvilinear Distance Analysis

While CCA is an important step towards dimensionality reduction of nonlinear manifolds, it still shares an essential limitation with other techniques like MDS. The use of the Euclidean distance on nonlinear manifolds yields misleading distance estimates between pairs of data points. To remedy for this drawback Lee et al. [LLDV00] proposed curvilinear distance analysis (CDA) as an extension of CCA. CDA replaces the Euclidean distance metric in the CCA algorithm by a *graph distance* metric (see Section 3.2.2). The graph distance metric is an approximation of the geodesic distance, which measures the distance between two points along the manifold. The graph distance is a much more faithful estimation of the pairwise distance than the Euclidean distance, and improves the ability to unfold highly nonlinear manifolds onto low-dimensional spaces.

Thus, CDA extends CCA in the same way Isomap extends classical MDS. The stress function optimized by CDA can be written similar to CCA as:

$$E_{CDA} = \sum_{i=1}^{N} \sum_{j=1}^{N} \left[\mathcal{D}^{\mathrm{G}}(\boldsymbol{x}(i), \boldsymbol{x}(j)) - \mathcal{D}^{\mathrm{G}}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j)) \right]^2 \; \mathcal{H}_{\beta(t)}(\mathcal{D}^{\mathrm{G}}(\widetilde{\boldsymbol{x}}(i), \widetilde{\boldsymbol{x}}(j))) \tag{3.26}$$

In contrast to CCA, however, the distance in the high-dimensional space is computed using the graph distance metric.

3.2.3. Comparison of Methods

This section compares the introduced DR methods according to different criteria. First, the techniques are classified according to their inherent characteristics. Next, the computational complexity of the algorithms is analyzed theoretically. This step is important for making informed choices about an appropriate DR algorithm when faced with a particular problem. As we will see, the computational complexity varies considerably among different algorithms. Finally, the algorithms are evaluated on a benchmark problem in order to assess the quality of the achieved embeddings.

Classification

By comparing the algorithmic details of the DR methods, we can observe, that some algorithms share common basic principles while others are founded on different theoretical grounds. In the following we will provide a set of (nonexhaustive) qualifications that allows us to classify and categorize DR methods.

DR techniques can be broadly divided into linear and nonlinear techniques. Linear techniques assume a linear relationship among the variables and are therefore, from a theoretical point of view, less powerful than nonlinear techniques. Nonlinear dimensionality reduction (NLDR) techniques can project data onto fewer dimensions, even if it lies on a complex, a nonlinear manifold.

Another difference between DR techniques is whether they are based on continuous or discrete models. Discrete techniques are based on a finite set of examples for which the DR problem is solved. This set is used as a discrete representation of the manifold. This leads to problems whenever a new point needs to be projected; the so called out-of-sample extension. In this case, an interpolation procedure is needed to combine the information from several points in order to interpolate the image coordinates of the new point. In contrast, continuous techniques can be seen as a parametrized function, which takes a given point as input and produces its image coordinates as output. This approach is more accurate than its discrete counterpart, which often introduces discontinuities into the projection and for which the question of accurate interpolation/extrapolation remains problematic.

Another important question is whether the DR process is based on exact or approximate optimization. Several algorithms, such as CCA or NLM, use stochastic gradient descent algorithms for computing the low-dimensional embedding. However, these algorithms are prone to convergence towards local optima. As a result, it is not guaranteed that the generated projection is globally optimal. The algorithm might generate different results for different consecutive runs. In contrast, methods based on some kind of eigendecomposition, so-called *spectral* methods, are guaranteed to converge towards the global optimum.

Finally, the DR techniques are often classified into distance or topology preserving methods. Distance preserving methods search for projections which retain the pairwise distances of the data set. By preserving these distances, the geometrical structure of the manifold can also be preserved. While this idea is intuitively correct, it leaves us with the problem of how to accurately measure distances along the manifold. Topology preserving methods try to retain the neighborhood relationships between points. Points that are near to each other in the high-dimensional space, should also be near to each other in the low-dimensional space, i.e. the exact distance does not play an important role. This allows topology preserving methods to deform a manifold to achieve a better embedding. However, such techniques are often based on a discrete representation of the neighborhood, which involves additional neighborhood parameters that often need to be supplied by the user.

Algorithm	Linearity	Preservation	Model	Solution
PCA	linear	-	continuous	exact
MDS	linear	distance	discrete	exact
LLE	nonlinear	topology	discrete	exact
Isomap	nonlinear	distance	discrete	exact
NLM	nonlinear	distance	discrete	approximate
CCA	nonlinear	distance	discrete	approximate
CDA	nonlinear	distance	discrete	approximate

Table 3.1: Classification of dimensionality reduction techniques into different categories based on their inherent characteristics.

Table 3.1 summarizes the results of the classification of the introduced DR techniques. Only LLE is a topology preserving method. Most other methods use a distance preserving approach. This can be explained by the fact that topology preservation is often difficult to specify in terms of a cost function. Often, prior knowledge about the approximate shape of the manifold is needed to create such a cost function.

Complexity

Analyzing the computational complexity of the DR techniques is important in order to assess their applicability in different domains, such as real-time environments. Both the computational complexity as well as the memory complexity can be important criteria on which to base the choice of DR technique. For most methods, the complexity is dependent upon the number of data points N, the dimensionality H of the original data, the dimensionality L of the embedding space, and in some cases the number K of nearest neighbors to take into account.

PCA performs an eigendecomposition on a centered covariance matrix. Here, the computational complexity of preparing the matrix amounts to $\mathcal{O}(HN)$. The eigendecompostion of the $H \times H$ matrix is performed in $\mathcal{O}(H^3)$. Figure 3.8 shows the computation times for the eigendecomposition for different numbers of data points. The plot confirms the theoretical analysis by empirical results and shows that the measured runtimes can be fit using a cubic function. Storing the eigenvector matrix has a memory complexity of $\mathcal{O}(H^2)$.

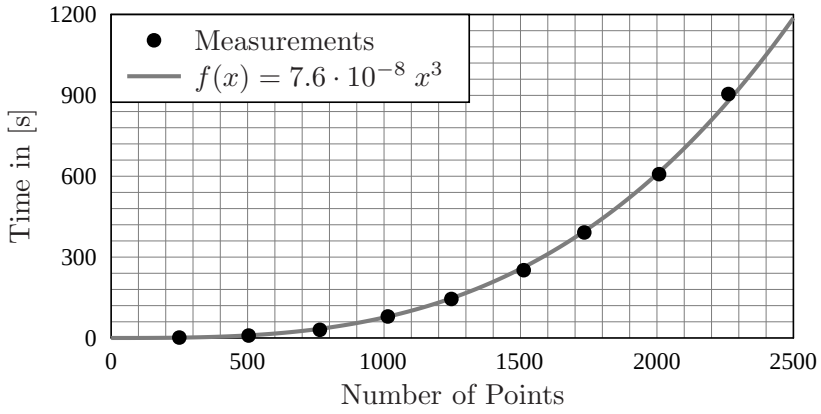


Figure 3.8: The time needed to perform an eigendecomposition for datasets with increasing size.

In contrast to PCA, MDS is much more computationally demanding and has a complexity of $\mathcal{O}(N^3)$. That is, for large datasets computing MDS becomes prohibitive. Similarly, Isomap also performs the eigendecomposition on a matrix of size $N \times N$ leading to a complexity of $\mathcal{O}(N^3)$. Additionally, Isomap performs Dijkstra's algorithm on the neighborhood graph in order to approximate the geodesic distances. This has complexity of $\mathcal{O}(NK + N\ \log\ N)$ using a Fibonacci heap implementation [FT87]. Furthermore, N nearest neighbor searches need to be performed with complexity $\mathcal{O}(HN\ \log\ N)$ [KR02]. Finally, because the full $N \times N$ matrix is stored, the memory consumption increases to $\mathcal{O}(N^2)$ compared to PCA. Taken together, the above considerations show that computation of MDS and Isomap can be very slow. To reduce the runtime of the algorithm, Isomap and MDS are often used in conjunction with preprocessing algorithms that se-

lect a representative subset of so-called *landmark* points [dST04]. For instance, Vector Quantization can be used for this. A faster variant of MDS can be found in [FL95].

LLE has the advantage of using a sparse matrix. As a result, the eigendecomposition has a computational complexity of $\mathcal{O}(rN^2)$, where r is the ratio of non-zero elements in the sparse matrix. During the embedding, LLE computes eigendecompositons for N matrices of size $K \times K$ with a complexity of $\mathcal{O}(NK^3)$.

Non-spectral methods such as CCA or NLM are based on iterative algorithms. Each iteration of the gradient descent optimization in such techniques involves $\mathcal{O}(LN^2)$ operations and memory space. While this is slower than PCA, it is in most cases faster than the computation of Isomap or MDS. This can easily be explained by the fact that we typically perform projections, where $L \ll N$. Therefore, as long as the number of iterations is smaller than $\frac{N}{L}$, non-spectral methods will be generally faster than Isomap.

Performance Benchmark

In the following experiment the DR techniques are applied to a data set sampled from the earlier introduced S-curve. The S-curve is a standard benchmark that is commonly used in the DR literature. The S-curve is an extruded S-shape, with a smooth, curved surface. The color coding is only used for easier visualization of the manifold and is irrelevant for the DR process. The task posed in this experiment, is to embed this surface in a two-dimensional space without loss of information. For this, the algorithms need to unfold the surface to a planar sheet.

Figure 3.9 shows the projections resulting from applying the different DR techniques to the benchmark problem. All algorithms were run on the same data set of 2500 points sampled from the S-curve benchmark. As can be seen, the results of PCA and MDS are not satisfying. In both cases we see superpositions of data points. More specifically, the red and blue parts of the manifold are intermixed with the green points. This effect can easily be explained, if we imagine the process of linear projection as taking a picture with a photo camera. Even if we vary the camera position, due to the shape of the S-curve, some points will always be in front of other points. A similar effect can also be seen for NLM. In contrast to that, the embedding performed by CCA appears more convincing. CCA succeeds to unfold the data for a large part of the manifold. Still, in particular at the boundaries of the manifold the embedding shows tearings. This introduces holes and outliers into the projection. LLE and Isomap produce satisfying projections, only slightly skewed or rounded at the boundaries. Finally, a perfect projection is achieved by CDA. The S-curve is perfectly unfolded onto a planar surface.

In Figure 3.10 the results reprojecting the image points back into the original three-dimensional space is shown. Confirming the above analysis, LLE, Isomap and CDA succeed in reconstructing the S-curve. This is mainly due to the fact that their projections did not introduce superpositions of points. In contrast to that, we see for MDS and NLM that in the upper (blue) and lower (red) part of the manifold several points collapsed. As a result, only the middle (green) part of the manifold can be reconstructed.

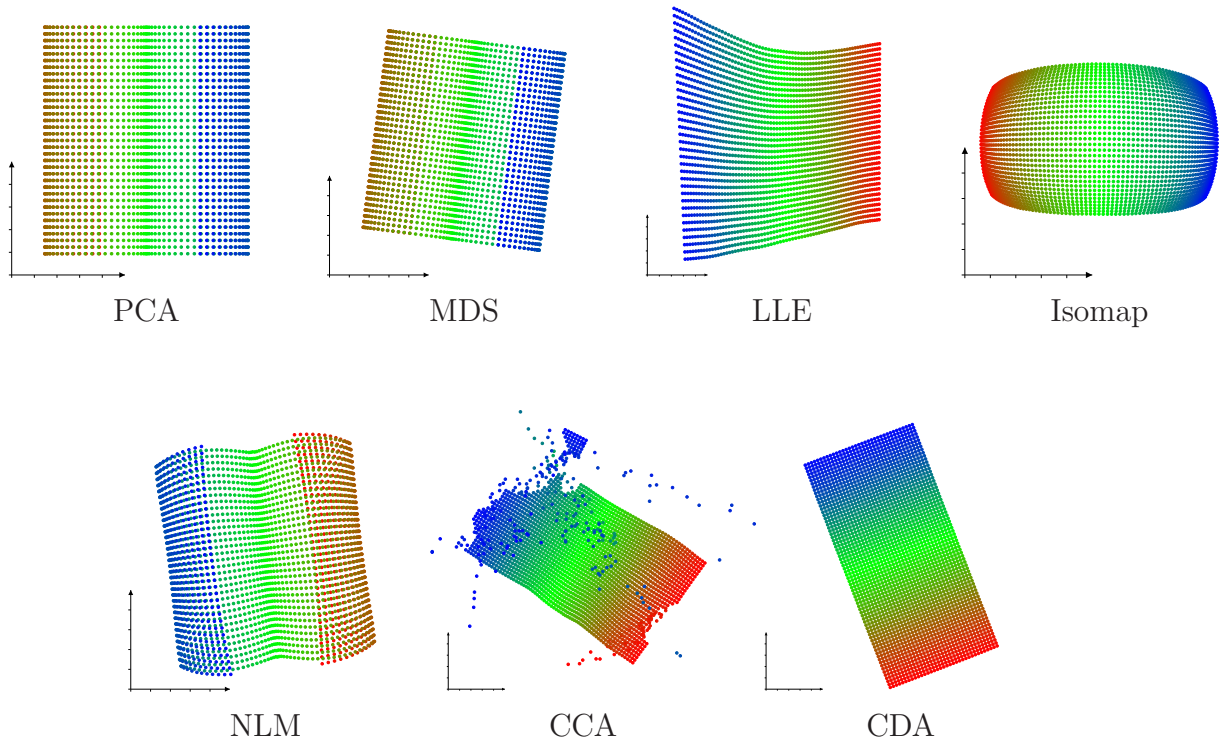


Figure 3.9: Projections of the S-curve on two dimensions performed using different dimensionality reduction techniques.

The worst reconstruction is exhibited by PCA, where all points are collapsed on a plane positioned in the center of the original S-curve.

3.2.4. Estimation of the Intrinsic Dimensionality

Almost all dimensionality reduction techniques described so far require the user to specify an input parameter L defining the dimensionality of the space on which to project the data. Whether the value chosen by the user is reasonable, or whether it is over- or underestimated is largely dependent on the data set and on the intended application. Still, in most cases the following question is critical: What is the smallest value for L, for which the data can be projected and reconstructed without loss of information? Logically, for all values of L lower than this threshold, the data can not be reconstructed without error. In the literature this threshold is called the *intrinsic dimensionality* of a data set. Loosely stated, intrinsic dimensionality refers to the minimum number of free variables needed to represent the data without information loss. In the case of the earth example, we can perfectly specify the position of each city by only two variables although the earth exists in three-dimensional space.

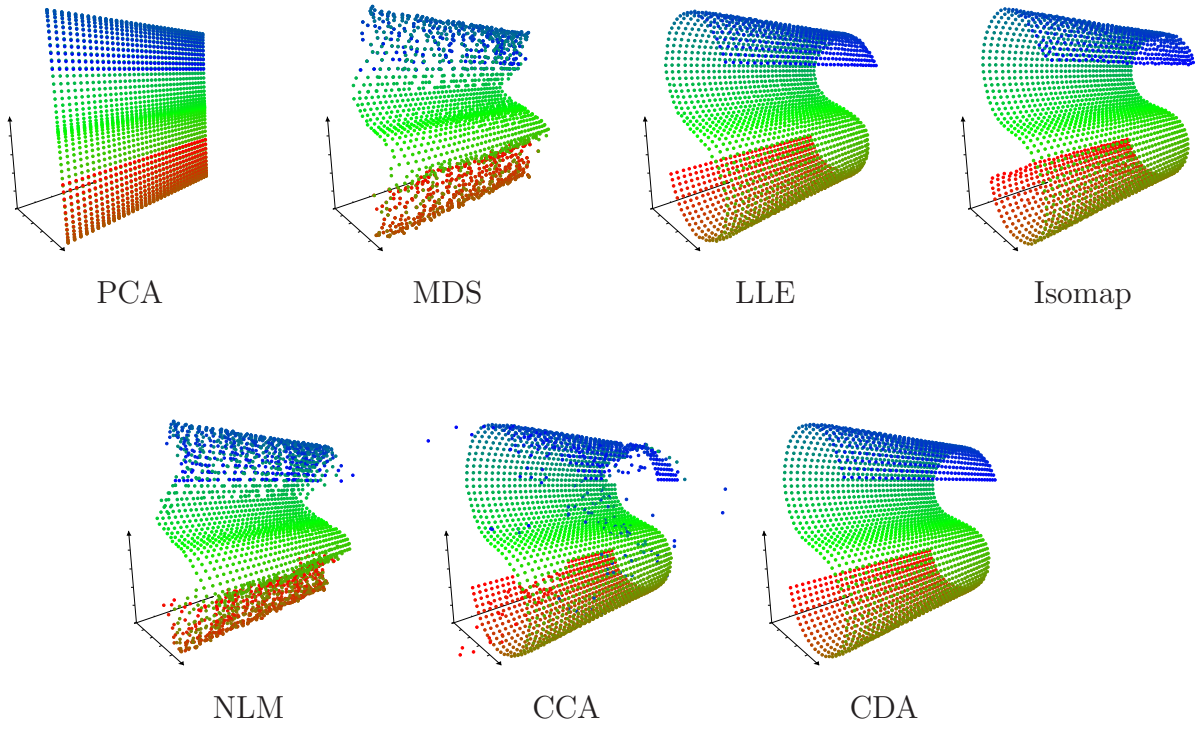


Figure 3.10: Reconstructions of the S-curve from the projected points.

Various methods exist for the estimation of the intrinsic dimensionality from a given data set. A simple way for achieving this, is the use of PCA. As already explained in Section 3.2.2, the eigenvalues contain information about the amount of information that is captured by each of the principal components. By computing Equation 3.9 for different values of L we obtain the percentage of the retained information as a function of L. Through the analysis of the resulting curve we can estimate the dimensionality of the data set. In many cases, the amount of retained information quickly converges towards 100%. The value of L for which this is the case can be used as an estimate of the intrinsic dimensionality. While this approach is a good starting point and generally gives good estimates, it still suffers from the fact that PCA is a purely linear technique. For nonlinear manifolds it is likely to overestimate the intrinsic dimensionality of the data.

The above limitation led to the development of *Local PCA*, which is arguably the most popular dimensionality estimation technique in the field of machine learning. Local PCA partitions the data set into small patches for each of which the dimensionality is estimated separately. The partitioning can be performed using a K-Means [Mac67] algorithm. The intrinsic dimensionality of the manifold is then computed by taking the average of all local estimations. The idea is similar to the approach followed by LLE: instead of processing the whole data set at once, divide it into locally linear patches and

process them separately. The underlying assumption is, that the investigated manifold is approximately linear on a local scale. In order to determine the dimensionality of a locally linear patch, Fukunaga et al. [FO71] introduced the $D\alpha$ criterion which regards an eigenvalue λ_i as significant if the following inequality holds:

$$\frac{\lambda_i}{\underset{j}{\text{argmax}}\ \lambda_j} \geq \alpha \tag{3.27}$$

where α is a given threshold parameter. In presence of noise the estimation of the intrinsic dimensionality often cannot be performed accurately. Still, the results of this process are often important in order to get an idea about the range of possible values and the ratio between the input dimensionality and intrinsic dimensionality of the dataset.

3.3. Conclusion

In this chapter we introduced important mathematical concepts that are vital for the imitation learning algorithm that will be developed and evaluated in this thesis. We first introduced hierarchical modeling of geometric structures, as well as different rotation representations that can be used to this end. Hierarchical modeling lies at the core of animating articulated structures.

We noted that, from a theoretical point of view, each of the discussed rotation representations has its advantages, as well as limitations. Therefore, a general recommendation as to which representation to choose can not be made. Instead, the rotation representation must be chosen with respect to the desired application domain.

In addition to hierarchical modeling, we also analyzed the mathematical concepts of dimensionality reduction. The analysis covered seven dimensionality reduction techniques, including both standard techniques and current state-of-the-art methods. The techniques greatly differ in their ability to uncover the structure of a given data set. A preliminary analysis of each of the techniques showed that nonlinear dimensionality reduction techniques can successfully project data onto lower dimensions without losing much of the intrinsic information. The analysis was, however, performed on a simple data set as used in other literature. Conclusions drawn from such experiments often do not generalize to other data sets or application domains. Therefore, in this thesis we will not prematurely decide in favor of a specific dimensionality reduction technique. Instead, in our experiments we will try to interchange the dimensionality reduction technique, in order to assess how the different techniques perform in our application domain.

4. An Imitation Learning Approach: Probabilistic Low-Dimensional Posture Models

4.1. Introduction

Imitation has a number of interesting qualities that make it appealing as a general model for programming synthetic humanoids. First of all, it is a fast way of transmitting knowledge between individuals. By watching and imitating the behavior of a teacher, we are spared of lengthy trial-and-error phases. Skills that may take weeks to learn by oneself, can be learned within days or hours in the presence of a teacher providing successful examples. Indeed, as stated in [BCDS08]: "(Imitation) is a powerful mechanism for reducing the complexity of search spaces for learning". That is, the observed demonstration can be used to rule out a large number of wrong attempts at mastering a skill.

From a computational point of view, imitation learning reflects a data-driven approach to the problem of motor skill acquisition. The traditional way of programming such skills is a labour and time intensive task, which requires a large amount of expert knowledge. In particular, it often involves the transformation of intuitive concepts of motions and actions into formal mathematical descriptions and algorithms. Even in GUI-based programming environments, where complex movements are specified as sequences of postures defined in the graphical user interface, much time is usually spent for parameter tweaking. Due to the relatively large number of degrees of freedom, this process becomes particularly cumbersome for the case of synthetic humanoids, such as androids or virtual humans. Imitation learning avoids these problems by stressing the importance of data. Information on how to perform a particular skill and what constraints to be taken into account, is directly extracted from recorded demonstrations. Hence, the focus of the user is shifted from programming to conveying 'meaningful' demonstrations. This is at the same time both a more natural and less labour intensive approach. While lay people are not familiar with intricate details of programming languages and AI, it is safe to assume that they have already been confronted with the situation of having to teach something to another person.

Two closely related features of imitation in which we are interested, are *adaptation* and *variation*. Adaptation refers to the ability to generalize a demonstrated posture or motion to an unseen context (see an example in Figure 4.1). Ideally, once a virtual human

Figure 4.1: Generalization of a learned 'box lifting' behavior. The joint angles are generated by taking into account the characteristics of the box (size, shape, position,..), as well as the anatomy of the synthetic humanoid.

learned to lift up a box, it should be able to reproduce this behavior, even if size and location of the box are changed. Variation refers to the ability to generate many variants of the same behavior. Human motion is by nature highly variable. Differences in motion can, for example, result from the genetic disposition, social and cultural background, personality or emotional state or simply the task at hand [Par07]. Even if we try to repeat a particular movement in the exact same way we are bound to fail. Natural noise on the level of muscle activations automatically leads to small perturbations in the joint configurations of the movement. Yet, such small-scale variations can have a strong impact on the believability of a humanoid character. Variation also takes place on a larger scale. The redundancy in human kinematics allows us to perform different realizations of the same motion. Therefore, adaptation and variation are interrelated properties of human motion.

Obviously, it is desirable to have a computational equivalent to natural imitation with above mentioned properties. Unfortunately, to what kind of observed behavior the term 'imitation' should refer, is still a matter of debate in the literature. This renders the concept vague and not well defined. In the following we will concentrate on what is called 'true imitation'—the acquisition of novel, previously unknown, motor skills. While biological plausibility is not our aim, we will build on various results in neuroscience and psychology. In particular, we adopt the concept of *behaviors*—primitives or building-blocks of motor control that can be combined to create a large repertoire of movements. A behavior is conceptually above the level of motor muscle commands and encodes complete temporal motor acts such as walking or dancing. There is substantial evidence that motor systems in humans and animals [BdST02] employ a modular organization based on a collection of primitives in order to generate a variety of movements. Many examples such as flocking have proved that the combination of seemingly simple behaviors can produce complex motion patterns. In neurophysiological studies with spinalized frogs it was found that stimulating different parts of the spinal

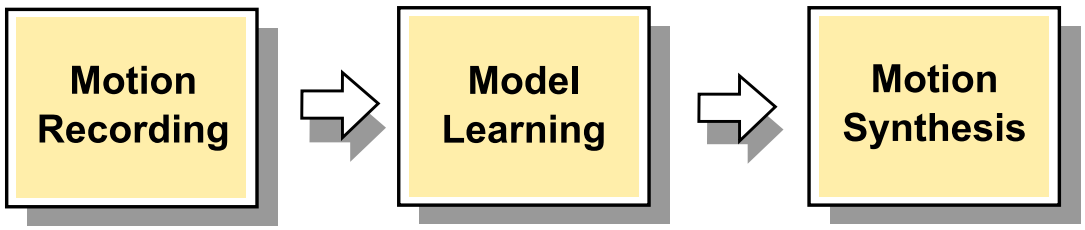


Figure 4.2: The three steps of our imitation learning approach: First, example motions are recorded using motion capture or other tracking devices. The acquired data is then processed and a model of thereof is learned. Finally, the model is used to synthesize variations of the recorded motions.

cord resulted in different movements of the frog's limb [MIGB94]. The forces acting on the limb varied based on the limb's position and on the stimulated spinal cord site. By fixing the spinal cord site and changing the position of the limb, it is possible to extract a force-field visualizing the forces encoded in a particular neuron group. Further investigation revealed that only a small number of distinct *force-field primitives* is needed in order to generate much of the frog's motor repertoire. Complex limb motion results from a linear superposition and sequencing of these primitives. Primitive behaviors can therefore be regarded as a vocabulary from which a rich set of motions can be composed [Mat02]. True imitation occurs, when a previously unseen behavior is learned and added to the motor repertoire. This makes behaviors the fundamental substrate of imitative capabilities.

Building on the above considerations we will now conceptualize a computational implementation of imitation learning and provide a general approach as well as its individual components. The presented imitation learning algorithm realizes 'true' imitation by learning single behaviors, which can then be chained to form complex sequences of actions.

4.2. The Three Steps in Imitation

Bakker and Kuniyoshi [BK96] identify the following three fundamental processes of imitation: (1) observation, (2) representation, and (3) reproduction. A student, aiming at imitating his teacher, first needs to observe one or several demonstrations. Then, he must create an internal model of the seen behavior. Finally, he needs to reproduce the behavior by himself, taking into account the current environment and situation. Our imitation learning approach adopts this view, although slightly reformulating the involved processes, to better fit the computational context. Hence, the approach consists of the three steps illustrated in Figure 4.2.

First, example motions from human teachers are recorded. This can be done using a variety of sensor technologies, such as motion capture systems, fingertracking, data

gloves or robot sensors. Once the example data is recorded, it is used to derive a computational model of the corresponding behavior. In particular, we are interested in generative models—models that can be used to synthesize variations of the demonstrated behavior. Finally, whenever the behavior is triggered, the model is used to synthesize the appropriate postures and motions taking into the account the context. These three steps are the basis of our algorithm, and will be executed whenever a new behavior is learned. The approach is flexible enough to be used with different realizations of synthetic humanoids, in particular robots and virtual characters. We will see in subsequent chapters, that this approach also supports different interpretations of imitation learning scenarios, including kinesthetic and close-contact interactions between the teacher and the student. To realize this approach the following components are needed:

1. **Search Space**: A space of behavior-specific postures derived from the recorded demonstrations. For example, a search space for a 'walking' behavior, should include a large number of typical walking poses.

2. **Task Constraints**: The set of constraints, in particular postural constraints, present in the behavior. While some of these constraints might stem from the task itself, we are also faced with anatomical limitations when dealing with synthetic humanoids. As the goal is to have a human-like appearance, it is important that the postures taken on by the agents appear realistic.

3. **Metric of Imitation**: The metric of imitation is an objective function which assigns a cost value to each attempt at reproduction of the demonstrated behavior. This is achieved by taking the current environmental situation into account. The metric allows us to evaluate and improve the agent's attempts at reproduction.

4. **Synthesis Algorithm**: An algorithm that takes the search space, the task constraints and the metric of imitation and produces a suitable variant of the learned behavior.

The first two components, i.e. the search space and the task constraints, will be derived using a novel learning method called probabilistic low-dimensional posture model (PLDPM). This method is the key element of the presented imitation learning method and is applied in the second step of our approach. The PLDPM construction algorithm automatically extracts correlations and synergies inherent to the demonstrated movements. These are then stored as a compact computational representation of the corresponding skill. In the final step, the posture synthesis step, an optimization algorithm is used in conjunction with the metric of imitation to synthesize a new posture or animation. Here, the metric of imitation acts as the objective function of the optimization process. In other words, the metric provides the optimization process with information about the quality of a given solution candidate. Using only this information, the optimization process explores the search space for a solution which best fits the criteria of the metric.

4.3. Probabilistic Low-Dimensional Posture Models

The PLDPM construction algorithm is an unsupervised learning method for generating a compact description of the kinematic and anatomic laws underlying a demonstrated behavior. It does so by exploiting the fact that natural motion can often be described as a combination of a small number of components. For instance, it was shown in [SFS98] using PCA that the first two principal components could account for more than eighty percent of the variance of grasping movements. Alexandrov and colleagues showed in [AFM98] that the first principal component captured more than ninety-eight percent of trunk bending movements. Similar results have also been reported for walking gaits [GBT04; SAD97], lip motion [RMGO96], catching movements [BTD10], typing [SF97], and smooth planar movements [San00]. In all cases the motions can be described by significantly fewer parameters than the total number of degrees of freedom. These findings are also supported by results from neurobiology and physiology. D'avella and colleagues [DB98] analyzed the frog hindlib response to vestibular stimulation and provided evidence that the muscular responses can be explained by a superposition of only five components.

Building on these findings, the PLDPM algorithm uses dimensionality reduction techniques to automatically extract a minimal set of principal components from exemplary kinematics data. This results in a low-dimensional posture space—a space in which each point represents a posture typical for the modeled behavior. Additionally, PLDPM also extracts the constraints, particularly anatomical constraints. These ensure that all synthesized postures or animations appear natural and realistic. Statistical methods are used to extract such constraints from the recorded data and represent them as probabilities. Hence, no prior knowledge about the mechanics of the modeled humanoid is needed. Formally, a PLDPM is defined as follows:

Definition 4 (Probabilistic Low-Dimensional Posture Model)**.** *Let set $\mathcal{Q} \subseteq \mathbb{R}^H$ be the set of recorded postures. The PLDPM of $\mathcal{Q}$ is a triplet $(\mathbb{P}, \Psi, \Phi)$, where:*

- $\mathbb{P} \subseteq \mathbb{R}^L$ *is a behavior-specific subspace of postures with* $L << H$,
- $\Psi : \mathbb{R}^H \rightarrow \mathbb{R}^L$ *is a function that returns for any vector* $\boldsymbol{q} \in \mathbb{R}^H$ *its low-dimensional image coordinates* $\widetilde{\boldsymbol{q}} \in \mathbb{P}$,
- $\Phi : \mathbb{R}^L \rightarrow [0..1]$ *is a probability density function that returns the probability of* $\widetilde{\boldsymbol{q}}$ *belonging to the same distribution as the projected example postures* $\widetilde{\mathcal{Q}}$.

The set $\mathcal{Q}$ is the training data of the algorithm and contains the recorded postures, i.e. joint angle values of an articulated structure. A posture $\boldsymbol{q} \in \mathcal{Q}$ can have any arbitrary rotation representation, such as matrix, quaternion or Euler representation. To extract the principle components from this data set, dimensionality reduction is applied. As introduced in Section 3.2 different methods can be used for this purpose. At the end of this step we have a posture space $\mathbb{P}$, the corresponding mapping function Ψ, as well

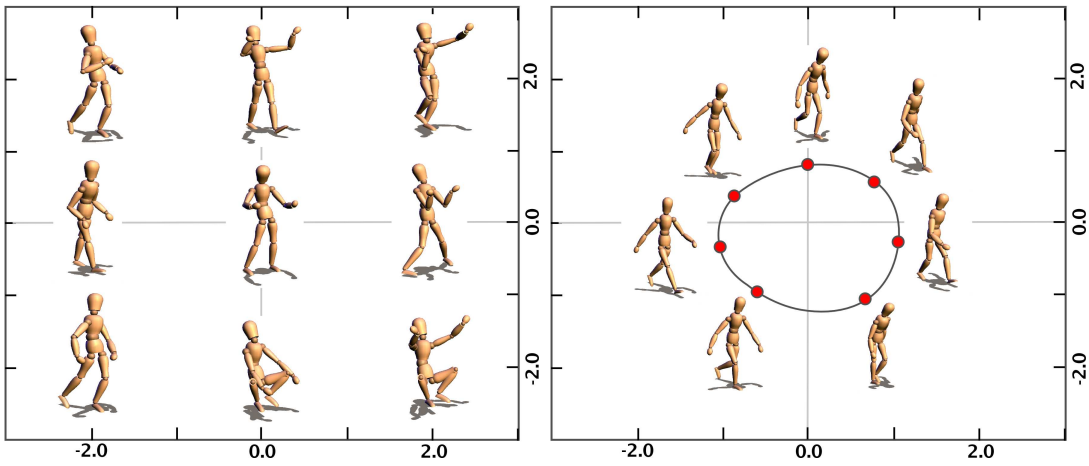


Figure 4.3: Left: A visualization of the low-dimensional posture space for box-lifting. Each point in the space corresponds to a posture. The postures of nine equidistant points are shown. Right: A visualization of the low-dimensional posture space for walking. The trajectory along the seven points specifies a walking motion.

as the projected images points $\widetilde{\mathcal{Q}}$. The function Ψ performs a mapping between the high-dimensional space of training data and the low-dimensional posture space. We can take any recorded posture $\boldsymbol{q}$ and determine its image coordinates $\widetilde{\boldsymbol{q}}$ by performing the operation $\Psi(\boldsymbol{q})$. In the other direction, we can take $\widetilde{\boldsymbol{q}} \in \mathbb{P}$ and reconstruct the joint rotations using $\Psi^{-1}(\widetilde{\boldsymbol{q}})$. Note, however, that dimensionality reduction is a kind of compression and typically leads to a loss of information. Therefore, in most cases the expression $\Psi^{-1}(\Psi(\boldsymbol{q})) = \boldsymbol{q}$ will not hold. Still, by an informed choice of parameter L, we may reduce the introduced reprojection error to arbitrarily low values.

The probability density function Φ models the task constraints, and gives important information about the feasability of a given posture. It is learned from the set $\widetilde{\mathcal{Q}}$ of image points. The rationale behing this, is that any new posture should obey the same distribution as the training data. Postures for which the image coordinates are far away from the vicinity of the training data, will most likely not meet the anatomical constraints underlying the behavior.

An important property of a PLDPM is that it takes a *discrete* set of postures and generates an infinite, *continuous* space of postures. Dimensionality reduction techniques are used to extract this space by uncovering the manifold underlying these postures. Based on a few recorded postures of a behavior, we can later generate an infinite number of interpolations and extrapolations thereof. This is rendered possible by the principle of *duality* between points in the extracted low-dimensional space and postures of the targeted synthetic humanoid. Any given point $\widetilde{\boldsymbol{q}} \in \mathbb{P}$ is fully sufficient to derive all kinematic parameters needed for specifying a posture $\boldsymbol{q} \in \mathbb{R}^H$ of the modeled humanoid. At the same time it holds, that for any posture $\boldsymbol{q}$ there exists a low-dimensional point $\widetilde{\boldsymbol{q}} \in \mathbb{P}$.

The principle of duality is illustrated in Figure 4.3 (left). A 2D posture space extracted from a 'box lifting' motion sequence is shown. The space is continuous, infinite and therefore we can generate an unlimited number of points therein. Due to the duality property, we can visualize each point in this space by reconstructing the corresponding joint rotations through $\Psi^{-1}(\widetilde{\boldsymbol{x}})$. In Figure 4.3 (left) this is done for nine different points in the posture space. Each of these points represents a different postural configuration. Yet, all of these postures are typical for 'box lifting' and correspond to different situations.

The concept of posture-point duality is also the basis for the generation of motions based on PLDPM. As can be seen in Figure 4.3 (right), an animation can be specified as a low-dimensional trajectory. The figure shows a 'walking' animation specified through seven control points in the posture space. Each control point represents a different key-frame of the animation. Because walking is a cyclical motion, the last control point is also the first control point; the postures at the beginning and the of a walking cycle are identical. The animation can now be replayed by reprojecting each point along the trajectory at different time steps. New animations or modifications of the original animation can be produced, by specifying a new set of control points.

The problem of animation synthesis can now be reformulated as follows: Given a learned PLDPM, find a low-dimensional point or trajectory, which optimizes a provided metric of imitation. The metric of imitation evaluates how well a given imitation attempt fits the current environmental context. The task of the synthesis algorithm is to explore the posture space using the values provided by the metric as a guidance. In other words, the task of posture synthesis is replaced by an optimization problem. Optimization theory provides a number of different techniques for solving similar problems. Our approach can be used in conjunction with any arbitrary optimization method.

In the remainder of this chapter, we will explain the details of the proposed imitation learning approach using a concrete example: the imitation of *'button pressing'*. This will help to easier understand the steps involved in the algorithm.

4.4. Example: Pressing a Button

In the following we will show how the introduced approach can be applied to imitate the way humans press a button. Without loss of generality, we will limit the analysis to the index finger.

The task of 'button-pressing' can be regarded as a simple goal-directed behavior, because its execution is dependent on the location of a target button. More precisely, we need to generate joint angles in a way, that at the end of the process the tip of the finger touches the surface of the button. For this, the position of the button needs to be taken into account when generating the motor signals. Before applying imitation learning to this task, let us first take a look at the kinematics and anatomy of the index finger.

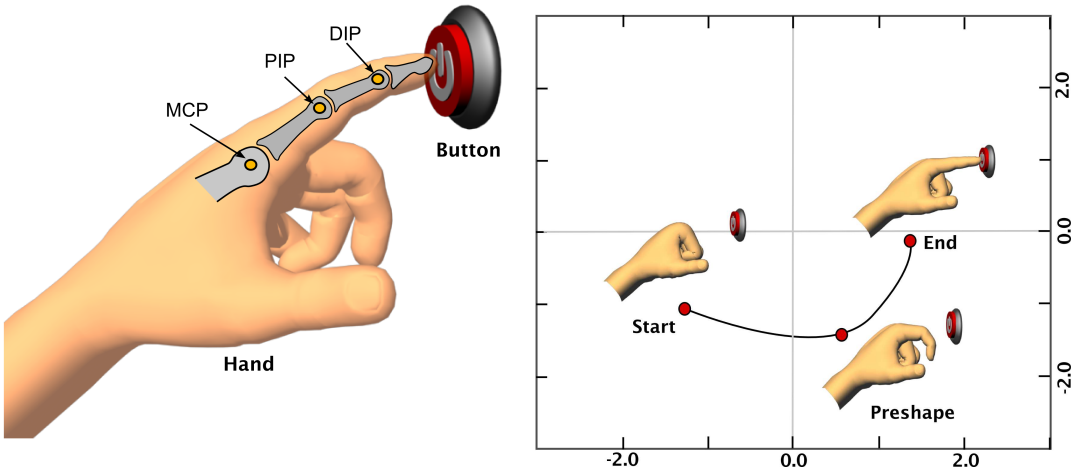


Figure 4.4: Left: Side view on the index finger's anatomy. Three joints realize the flexion, extension, abduction and adduction movements of the finger. Right: Low-dimensional trajectory going through three key postures.

The skeleton of the index finger can be modeled as an open kinematic chain of three rigid bodies: the proximal phalanx, the middle phalanx, and the distal phalanx. The behavior of the kinematic chain is determined by three joints, namely the matacarpal-phalangeal (MCP), the proximal-interphalangeal (PIP), and the distal-interphalangeal (DIP) joint, see Figure 4.4 (left). The two intephalangeal joints PIP and DIP are often described as hinge joints having one DOF. They are responsible for the flexion/extension movement of the finger. The MCP joint has a second DOF which allows for adduction (movement towards the middle finger) and abduction (movement away from the middle finger). The finger joints are highly interdependent, as the activation of one of the controlling muscles will typically result in a coactivtion of another muscle. Trying to capture this interdependence by mathematical means, Rijkema and Girard [RG91] related the joint angle of the distal- to the proximal-interphalangeal joint by the equation $\theta_{DIP} = \frac{2}{3}\theta_{PIP}$. However, in [ES03] it was reported that this approximation is too coarse for intricate control of the hand. Generally, such models can only be used as crude approximations of the joint interdependencies, due to the large range of anatomical and biomechanical properties which can be found in different humans. Other important finger parameters that are difficult to model are the joint limits. Depending on the joint laxity of a person, the finger can flex and extend to a different degree. Due to genetic dispostion or special training, such as sports or yoga, some people are more flexible than others. A general rule describing the joint limits is therefore bound to fail.

Using the introduced PLDPM method, we will model the 'button-pressing' behavior as a trajectory in a low-dimensional posture space. This space will be learned from recorded example data. A visualization of a possible trajectory is given in Figure 4.4 (right). The behavior can be specified as a trajectory through three control points in the posture space.

The starting point represents a closed hand. The second control point, the *'preshape'*, corresponds to an extended index finger. Finally, the last control point corresponds to a hand configuration where the finger touches the button. In order to produce successful realizations of the behavior, our synthesis algorithm needs to ensure that the index finger always touches the button at the end of the animation. The metric of imitation is, therefore, based on the distance between the tip of the finger and the button. Ideally, the distance is zero, that is, the finger tip touches the surface of the button.

4.4.1. Motion Recording

The first step of creating a PLDPM for finger movements is the collection of example data. For this, we used a fingertracking device capturing a test-subject's hand movements. Other recording technologies can also be used for this purpose.

During movement recording, a subject was asked to move its index finger for about 5 minutes and touch different buttons. The finger configuration was repeatedly captured and stored as vectors of joint rotations. The rotations were represented as Euler angles, thus, leading to a vector with 9 elements (3 angles $\times$ 3 joints). Note that due to our expert knowledge about the finger anatomy, we could have reduced the number of entries in the posture vector. We know that both the rotation of the PIP and DIP has only one DOF and can therefore be represented using one value. However, we avoided using any kind of expert knowledge in the design of the experiment. Note also that the storage of the rotations in a Euler representation it is not mandatory. As we shall see in later experiments, it can be beneficial to use other types of rotation representations.

After recording, we performed vector quantization as a first preprocessing step on the gathered data. Vector quantization deals with the problem of how to encode a large set of data vectors using a smaller, but still representative set of *prototype* vectors. This reduces the number of data samples in the data set, while retaining the structure of the manifold.

The reason for this step was already explained in Figure 3.8, which shows the time needed to derive the matrix of eigenvectors as a function of the number of samples in the training set. The computation of the eigenvectors is the core mathematical operation for most dimensionality reduction techniques. There is a cubic relation between the number of points and the needed computation time. With increasing number of points, the derivation of the eigenvector matrix becomes more and more demanding and time consuming. Therefore, whenever the dataset of training data is large, it is helpful to first apply vector quantization.

A classical vector quantization algorithm is the K-means algorithm [Mac67], which uses an iterative procedure in which two successive steps are repeated to find ideal prototype vectors. Given the number of clusters as a parameter, the K-means algorithm starts by (randomly) initializing the prototype vectors. Afterwards, the iterative two-step algorithm starts. In the *assignment* phase, each data vector is assigned to the nearest prototype vector, yielding a Voronoi partition of the original data set. In the

update step, the prototype vectors are updated, by computing the mean (center of mass) of the points associated with it. The algorithm terminates when the protoypes stay fixed during an iteration step.

To run the K-means algorithm, the number of clusters must be provided by the user. However, the ideal number of clusters is heavily dependent on the data set at hand. If this number is too small, the quantization error will be large. Conversely, if too many clusters are allocated, the efficiency of quantization will be reduced, because some data clusters will have two prototypes. Instead of fixing the number of prototypes, we use a dynamic vector quantization algorithm.

In particular, the *Neural Water* algorithm [LH99] is used, which introduces an additional parameter r specifying the radius-of-influence of each prototype. In K-means, each prototype was influenced by all points in its Voronoi region, i.e. to which it is closest. In contrast to that, in Neural Water each prototype is only influenced by points inside the hypersphere of radius r. If a data vector is not inside a hypersphere of any of the prototypes, then a new prototype is created. The radius-of-influence parameter is computed based on the maximal distance `maxdist` between any two data vectors in the original data set. In our experiments we always initialized it to $r = 0.06 \times \texttt{maxdist}$.

4.4.2. Model Learning

Before learning a probabilistic low-dimensional posture model, we need to find out how many DOF we shall retain in the data set. In other words: On how many dimensions do we have to project the data? This question can be answered by estimating the intrinsic dimensionality. In Figure 4.5 (left) we see a graph showing the results of dimensionality estimation for the index finger data set.

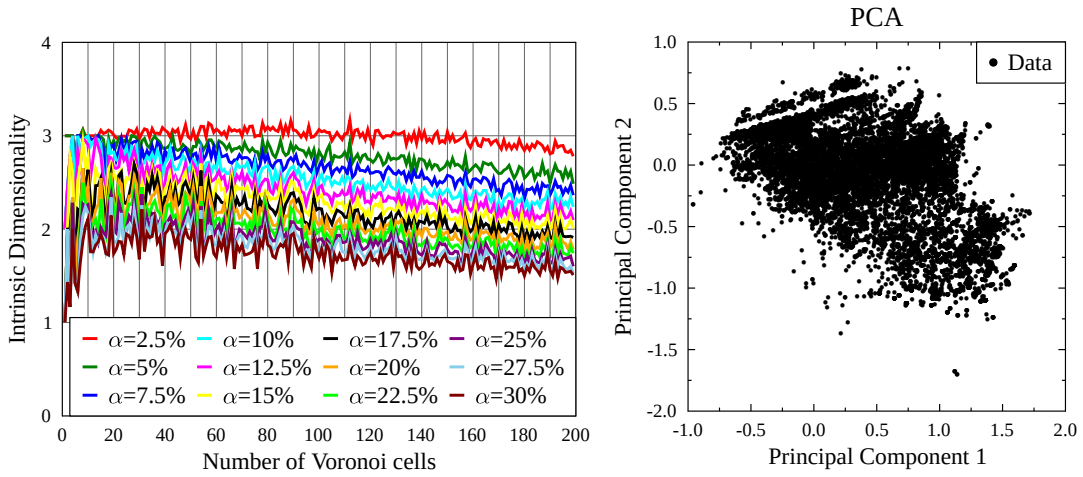


Figure 4.5: Left: Intrinsic dimensionality analysis of a data set. The intrinsic dimensionality is estimated depending on the number of Voronoi cells employed and the used α value. Right: Projection of the data on two dimensions using PCA.

The plot shows the estimated dimensionality for different values of $D\alpha$ and different numbers of Voronoi cells. Most estimates are in the range between two- and three dimensions. However, with increasing number of Voronoi cells, most estimates slowly converge towards two dimensions. This result is congruent with our earlier discussion and confirms the fact that movement of fingersegments is highly interdependent. Yet, it is remarkable that all our index finger movements, even the complex and intricate ones, can be expressed with at most three variables. Based on this result, we can make an informed decision on the number of dimensions on which to project the data. This often involves a tradeoff between model complexity and desired reconstruction error. If we have large values for the number L of dimensions, then the model retains its high dimensionality and the effect of DR will be mininal. On the other hand, if we choose too low values for L, then much information will be lost during the reduction process.

For our index finger example we will project the data into a 2D space. This is done by applying any of the algorithms introduced in Section 3.2 to the set of training data. The output of this process is the function Ψ which allows us to project any point into the two-dimensional posture space and, conversely, any point in the posture space back into the original domain. To analyze the amount of error introduced through this process, we projected all training data into the posture space and vice versa and measured the difference between the original and reprojected vectors. Table 4.1 presents the sum of errors over all joints in Euler angles. For example, the sum of errors in joint angles for PCA with one dimension is 10.047°, which translates to an error of approximately 3° per joint.

Dim.	PCA	LLE	CCA	CDA	MDS	ISOMAP	NLM
1	10.047	8.71949	36.7324	12.6588	8.96973	9.03659	8.89414
2	7.41234	5.68799	39.0204	42.2875	5.9788	8.17665	5.72388
3	5.65845	4.77577	75.138	48.6387	4.56159	5.49271	4.09325
4	1.58535	3.45906	77.7155	66.9494	0.798947	4.72576	0.788495
5	0.296043	2.03171	90.1142	80.0184	0.779449	4.38621	0.779471
6	0.296043	3.27663	67.4426	69.4863	0.778885	4.12079	0.779471
7	0.296053	1.3857	67.2208	76.7819	0.778842	4.0023	0.779471
8	0.296053	3.92034	61.6428	58.1566	0.778768	3.66253	0.779471
9	0.296053	3.63381	54.5828	58.9622	0.778663	2.79237	0.779471

Table 4.1: The reprojection error introduced by using different dimensions of the low-dimensional space and different DR techniques.

Analyzing the results of Table 4.1 we find that LLE, NLM and MDS achieve the best results among the DR techniques investigated. The worst performance is due to CCA. PCA has a slow start with mediocre reprojection error for the first four dimensions. At the same time we notice that, beginning from dimension five, PCA has the lowest repro-

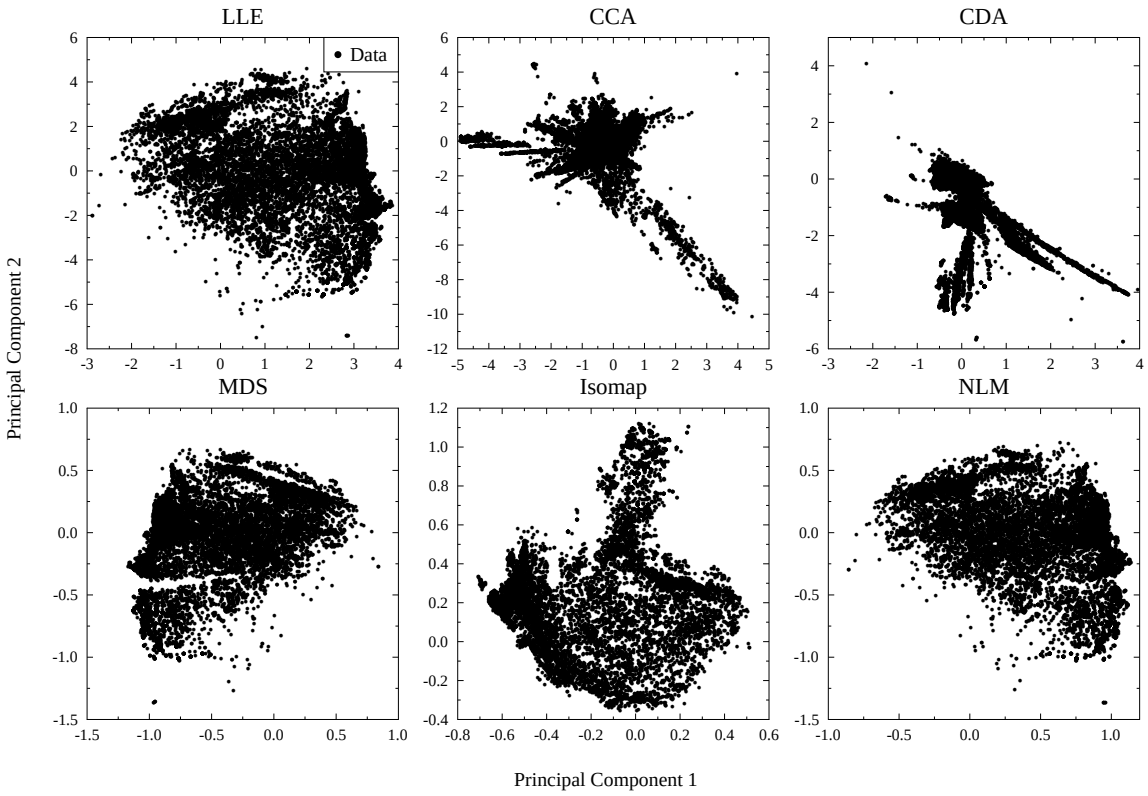


Figure 4.6: Projection of the finger kinematics data using different dimensionality reduction techniques.

jection error of all techniques. Further, this value remains unsurpassed by the results of other techniques even with a comparably higher number of dimensions. Obviously, various other techniques are better than PCA in compressing the data using fewer dimensions. On the other hand, PCA is the method of choice whenever the original data needs to be closely reconstructed.

Figure 4.6 shows the projected training data in different low-dimensional spaces. The projections explain why LLE, NLM and MDS achieve similar results. Apart from the range of values, the results of LLE and NLM appear nearly identical. The MDS result also resembles these projections, only mirrored. In contrast, the results of CCA and CDA are in no way comparable to the results of any other technique. Instead of a data cloud the set of projected points take slightly star shaped form with long streaks of outliers in different directions. This is a strong indicator for an unsuccessful embedding, given that the data was captured from continuous finger motions with only slight angular differences between successive frames. This hypothesis is also backed by the results in Table 4.1. Both CCA and its variant CDA are the two worst performing techniques in this scenario. The reason for this result can be traced back to the neighborhood function

employed by both techniques. It allows us to focus on the preservation of distances on a given scale only and is generally an important tool for unfolding manifolds with complex shapes. In some cases however, in particular in the presence of noise, it might introduce some unnecessary tearings of the data.

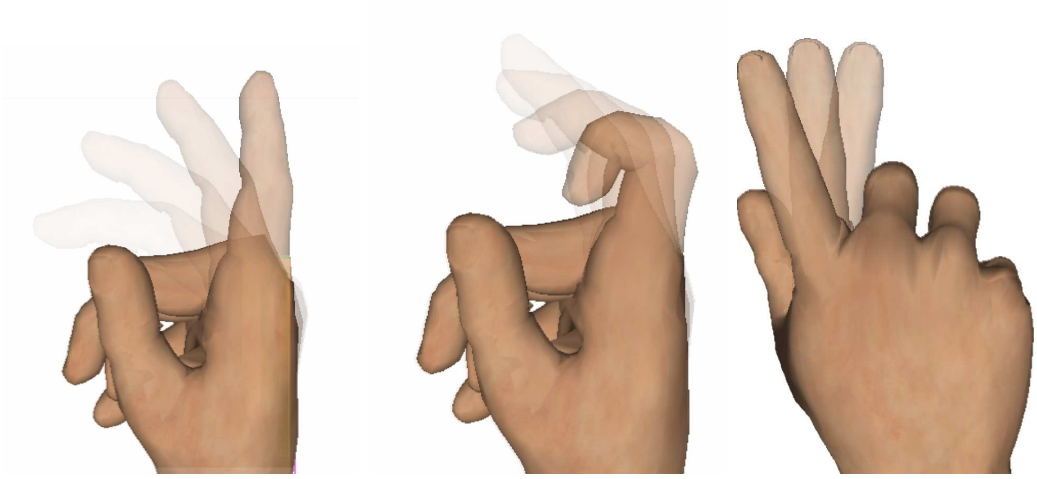


Figure 4.7: The first three principal components extracted using PCA. Left: The first principal component stores the rotation around the MCP joint. Middle: The second PC encodes the flexion and extension of the finger. Right: The third PC encodes the abduction and adduction movements of the finger.

Figure 4.7 visualizes the first three principal components extracted via PCA. The components correspond to a rotation around the MCP, flexion/extension, and abduction/adduction. This shows the interesting feature of PCA of extracting *'meaningful'* components, i.e. components that can be linked to existing concepts, for example in biomechanics. Nonlinear DR techniques on the other hand extract components that are difficult to analyze and which often cannot be categorized into existing traits or concepts. Here, one component often combines different tasks. While this renders human analysis difficult, it results in higher compression rates of the data. In our particular example, a principal component extracted via NLDR techniques might encode *both* flexion/extension and abduction/adduction to some extent.

Modeling Anatomical Constraints

At this stage, we have a set of image points in a two-dimensional posture space. The projections do not cover the whole posture space. Instead, we find the projections to be distributed in particular areas or clusters in the posture space. This is due to the anatomical limitations of the human body. Not all postures that are reflected in the posture space are anatomically plausible and reproducable by a real human. The joint limits strictly constrain the human body to particular types of postures. To uncover and model these constraints, PLDPM employs probability density modeling. Concretely, a

Gaussian mixture model (GMM) is used to create a statistical model of the distribution of recorded finger motions in the posture space. Regions with a high density of projected points will be assigned to a high probability. On the other hand, regions with few projected points will receive a low probability. The resulting probability density function can be regarded as a data-driven approximation of the anatomical plausibility of finger postures. During synthesis, the statistical model can be queried for the probability of a given finger posture. Hence, postures with low anatomical plausibility can be avoided.

The main problem in density estimation can be formulated as follows: Given an arbitrary set of data vectors with no or little prior knowledge of their structure, how can we devise a probability density function which best fits this data set? One possible solution to this problem is to fit a Gaussian distribution to the data. However, if the data set is not normally distributed, the resulting fit would give a poor description of the data. This problem can be overcome, by using a mixture of Gaussian distributions. The rationale behind this approach is that any continuous density function can be approximated with arbitrary accuracy using a superposition of Gaussian distributions. The resulting model is therefore called a Gaussian mixture model. Given all data points, the GMM estimates a probability density function by a weighted sum of K Gaussian distributions. The probability density function can be written as:

$$\Phi(\boldsymbol{x}) = \sum_{k=1}^{K} \boldsymbol{\pi}_k \, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k) \tag{4.1}$$

with $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k)$ being the k-th Gaussian distribution and $\boldsymbol{\pi}_k$ being the corresponding weight. The Gaussian distribution has dimensionality L and is defined by:

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k) = \frac{1}{\sqrt{2\pi}^L \sqrt{det(\Sigma_k)}} \, \mathrm{e}^{-\frac{1}{2}\left((\boldsymbol{x}-\boldsymbol{\mu}_k)^T \; \Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right)} \tag{4.2}$$

with mean $\boldsymbol{\mu}_k$ and covariance matrix Σ_k. The parameters $\{\boldsymbol{\mu}_k, \Sigma_k, \boldsymbol{\pi}_k\}$ can be estimated using a maximum likelihood estimation. For this, we need to maximize the likelihood of the data given the model parametrized by $\{\boldsymbol{\mu}_k, \Sigma_k, \boldsymbol{\pi}_k\}$. The logarithm of the likelihood function of Equation 4.1 can be written as:

$$\sum_{n=1}^{N} \ln \left[\sum_{k=1}^{K} \boldsymbol{\pi}_k \; (\boldsymbol{x}(n)|\boldsymbol{\mu}_k, \Sigma_k) \right] \tag{4.3}$$

To maximize the log-likelihood we can use the expectation-maximization (EM) algorithm [DLR77]. EM estimates the parameters of a GMM using an iterative scheme involving two steps. In the first step, called the *expectation step*, the posterior probability for each of the kernels is evaluated. The posterior probability can be regarded as the probability that a particular kernel k produces the observed data. This is sometimes

also referred to as *responsibility*: the responsibility of kernel k in explaining the observed data. The estimation is done according to the following formula:

$$\Phi(k|n) = \frac{\boldsymbol{\pi}_k \, \mathcal{N}(\boldsymbol{x}(n)|\boldsymbol{\mu}_k, \Sigma_k)}{\sum_{i=1}^{K} \boldsymbol{\pi}_i \, \mathcal{N}(\boldsymbol{x}(n)|\boldsymbol{\mu}_i, \Sigma_i)} \tag{4.4}$$

The posterior probabilities are then used in the maximization step to reestimate the parameters of the GMM:

$$\boldsymbol{\pi}_k \longleftarrow \frac{\sum_{n=1}^{N} \Phi(k|n)}{N} \tag{4.5}$$

$$\boldsymbol{\mu}_k \longleftarrow \frac{\sum_{n=1}^{N} \Phi(k|n) \; \boldsymbol{x}(n)}{\sum_{n=1}^{N} \Phi(k|n)} \tag{4.6}$$

$$\Sigma_k \longleftarrow \sum_{n=1}^{N} \Phi(k|n) \; \frac{(\boldsymbol{x}(n) - \boldsymbol{\mu}_k) \; (\boldsymbol{x}(n) - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^{N} \Phi(k|n)} \tag{4.7}$$

An important theoretical result of Dempster et al. [DLR77] is that with each iteration of the above two steps, the log-likelihood is guaranteed to increase. This leaves us with a simple iterative procedure for the maximization of the log-likelihood, in which the expectation and maximization steps are repeatedly performed until the parameters do not change. Unfortunately, the algorithm is only guaranteed to converge towards a local optimum of the likelihood function. Similarly to K-means, the application of the algorithm with varying initial values leads to different results. Therefore, in application the algorithm is run various times and the result with the highest log-likelihood is then used.

The K-means algorithm and the GMM algorithm are closely related. In fact, it can be shown that the K-means algorithm is a special case of the GMM algorithm. The EM algorithm is computationally much more demanding and also with respect to the number of iterations slower than the K-means algorithm. Additionally, GMMs have problems dealing with high-dimensional data sets and their use is often limited to data sets with few dimensions. The estimation of the parameter K is, similarly as in K-means, a non-trivial question. A common approach to solve this problem is to compute a set of GMMs with increasing values for K. Afterwards, the Bayesian information criterion (BIC) can be used to evaluate each model according to how well it fits the training data and how complex it is. This approach can be computationally demanding. A more efficient approach is to first apply a K-means algorithm on the data before executing the GMM algorithm.

In Figure 4.8 the process of estimating a probability density function from a set of points projected via the Isomap algorithm is shown. Using the Neural Water algorithm,

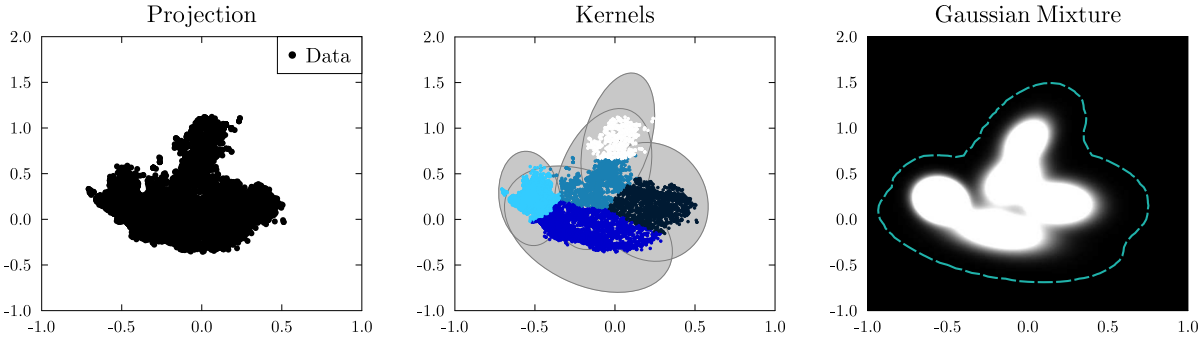


Figure 4.8: Learning a Gaussian mixture model for the projected finger kinematics. Left: Projected data points. Middle: Five Gaussian kernels are fitted to the data set using the EM algorithm. Right: A grayscale representation of the encoded probabilities of the posture space.

a number of Gaussian kernels are fit to the data. In Figure 4.8 (middle) each point is colored based on the kernel to which it belongs. The result is then processed by a GMM, leading to a probability density function Φ, visualized in Figure 4.8 (right). Here, each point $\widetilde{\boldsymbol{x}}$ in the posture space is assigned a gray value according to the probability $\Phi(\widetilde{\boldsymbol{x}})$. White areas correspond to spots of high probability, and therefore high anatomical plausibility. Dark areas correspond to postures of low probability. The dashed blue curve in Figure 4.8 (right) encloses the area where all points have a probability which suffices the inequality $\Phi(\widetilde{\boldsymbol{x}}) > \tau$, where $\tau = 0.0001$. In the synthesis algorithm which will be discussed in Section 4.4.3, all postures corresponding to points outside this area will be discarded.

4.4.3. Posture Synthesis

We can use the learned PLDPM to synthesize new postures and motions for the index finger using optimization techniques. In the following we will introduce the basic concepts of optimization and different types of optimization algorithms, namely gradient descent, hill climbing, and evolutionary algorithms. Then, we will explain in detail how optimization is used for imitation.

Theory of Optimization

Optimization is a search process for the optimal parameters of a system with respect to a given criterion, which is called the *objective function* of the optimization process. The objective function evaluates a particular candidate solution and assigns a numerical value to it. This numerical value indicates the *'score'* or *'goodness'* of the evaluated solution.

An objective function $f : \Omega \rightarrow \mathbb{R}$ is a mathematical function which is subject to optimization. From an optimization point of view, the concept of *'metric of imitation'* introduced earlier is just an objective function. Optimization algorithms try to find solutions in the search space Ω that lead to an optimal value of the objective function. Accordingly, an optimization problem is defined as:

Definition 5 (Optimization Problem)**.** *An optimization problem is a tuple* $O = (\Omega, f, \succeq)$ *where* Ω *is the search space,* $f : \Omega \rightarrow \mathbb{R}$ *an objective function and "* $\succeq$*"* $\in \{\leq, \geq\}$ *a comparison operator defined on* $\mathbb{R}$*. The goal is to find a point* $\boldsymbol{x} \in \Omega$ *which is a global optimum of* f*.*

The comparison operator is used in order to compare the score of two points in the search space. The inequation $f(\boldsymbol{x}) \succeq f(\boldsymbol{x}')$ means: the score of $\boldsymbol{x}$ is higher or equal to the score of $\boldsymbol{x}'$. The goal of a *global optimizer* is to find the global optimum of the objective function f.

Definition 6 (Global Optimum)**.** *A point* $\boldsymbol{x} \in \Omega$ *of the search space is called a global optimum with respect to* $\succeq$ *if,*

$$\forall \boldsymbol{x}' \in \Omega : f(\boldsymbol{x}) \succeq f(\boldsymbol{x}') \tag{4.8}$$

From this definition follows that a global optimum is the point $\boldsymbol{x}$ with the highest objective score among all points in the search space. In contrast, a local optimum only needs to have the highest score among all points within a vicinity specified by ε.

Definition 7 (Local Optimum)**.** *A point* $\boldsymbol{x} \in \Omega$ *of the search space is called a local optimum if,*

$$\exists \varepsilon \in \mathbb{R}, \varepsilon > 0, \forall \boldsymbol{x}' \in \Omega : ||\boldsymbol{x} - \boldsymbol{x}'|| < \varepsilon \Rightarrow f(x) \succeq f(x') \tag{4.9}$$

Each global optimum is also a local optimum. The inverse statement does not hold.

Depending on the point of view and the terminology, optimization can be formulated as a minimization or a maximization problem. If the result of expression $f(\boldsymbol{x})$ reflects the cost of using solution $\boldsymbol{x}$, then optimization is cast as a minimization problem. In contrast, if the results of above expression reflect the utility of solution $\boldsymbol{x}$, then the problem is formulated as a maximization problem.

Optimization Algorithms

A naive way of optimizing a given function, is to perform an exhaustive search over its parameters. All possible parameter settings are tried and the corresponding objective values are recorded. At the end, the setting which has the highest objective value is selected. Although this approach ensures that no optimum is overlooked, it is, even for relatively simple problems, impractical. Modern optimization algorithms try to optimize a function by using a minimal number of iterations. This can be achieved in different

ways. In the following, we will introduce three families of optimization algorithms and discuss their characteristics.

Gradient Descent

The *gradient descent* algorithm is a fundamental method of optimization and the basis of many machine learning algorithms. The general idea is to determine the direction of the maximum rate of increase at a starting point $\boldsymbol{x}$, and then walk into the opposite direction until a minimum of the function is found. The direction of maximum rate change of function f at a point $\boldsymbol{x}$ is determined by computing the gradient $\nabla f(x)$. By following the negative gradient, we iteratively go downhill in the direction of a (local) minimum. Algorithm 2 presents a generic gradient descent algorithm.

Algorithm 2 Gradient descent optimization algorithm with starting point $\boldsymbol{x}$, step size η, and threshold θ

Require: $\boldsymbol{x}, \eta, \theta$
1: **while** $|\nabla f(\boldsymbol{x})| > \theta$ **do**
2: Calculate the gradient vector $\nabla f(\boldsymbol{x})$
3: $\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \nabla f(\boldsymbol{x})$
4: **end while**

The parameter η denotes the step size of the algorithm. In the machine learning literature this parameter is often called the *learning rate*. As the name suggests the parameter defines the size of the steps with which the algorithm proceeds downhill at each iteration. A large step size results in fast descent towards the minimum, but also bears the risk of overshooting the minimum. In this case the algorithm never finds the optimum, because it always passes it which leads to an oscillatory search behavior. On the other hand, a small step size trades the increased probability of finding the exact value for the minimum for a slower convergence rate. The search is repeated until the gradient is smaller than a user-supplied threshold θ.

The step size has a strong impact on the performance of the algorithm and is in practice often difficult to choose. Over the years many suggestions have been made in order to determine good values for η. However, experience has shown that most of these approaches do not lead to a general improvement of the algorithm performance. This can be explained by the fact that the size of the update step does not only depend on the step size parameter, but also on the partial derivative. Therefore, even if an appropriate value for η is found, the effect can be "drastically disturbed by the unforeseeable behavior of the derivative itself"[RB93]. This problem was observed by Riedmiller and Braun and led to the development of the resilient backpropagation (RPROP) algorithm [RB93]. The RPROP algorithm is an adaptive optimization technique developed in the context of Neural Network learning. Adaptive means, that the parameters, in this case the step size, are changed at runtime.

Other gradient-based optimization techniques are, for example, the conjugate gradient method, the Newton method, or the Broyden-Fletscher-Goldfarb-Shanno (BFGS) method. The Newton method is a second-order technique which uses not only the gradient, but also the secondary derivatives, represented by a Hessian matrix. However, this comes at the price of storing and inverting the Hessian matrix, which is a computationally expensive task with a quadratic time and space complexity. The BFGS algorithm tries to circumvent this problem by estimating the Hessian matrix from successive gradient vectors.

In [IH03] an empirical evaluation was performed, comparing variants of RPROP to BFGS, conjugate gradient and other algorithms in the domain of machine learning. Despite being a first-order method, it was shown that RPROP outperforms both BFGS and conjugate gradient in that domain. Additionally, RPROP has elegant features like linear time and space complexity, robustness with respect to internal parameters, robustness with respect to numerical error[1], and an easy implementation.

Note, that gradient-based algorithms require the computation of the gradient and assume a continuously differentiable objective function f. Of course, these problems are interconnected. Given that the function is continuously differentiable, we can numerically compute the gradient by finite differences. However, this requires additional evaluations of the objective function. Finite-difference approximation is also susceptible to truncation and round-off errors and is therefore of limited accuracy. A more accurate and efficient way to compute the gradient is to use automatic differentiation [GC91]. Automatic differentiation techniques perform differentiation on code-level. In other words, an objective function given as source code in a certain programming language is object to the derivation procedure. By applying a set of transformations such as the chain rule, the source code for computing the objective function is transformed into the source code for computing the derivative thereof. In contrast to finite-differences, results of high-accuracy can be achieved by these techniques. However, this still requires the objective function to be continuously differentiable, which is often not the case. Indeed, most objective functions that have been investigated during the writing of this thesis do not fulfill this requirement. A prominent approach to tackle this problem is to use stochastic optimization methods which we will discuss in the following.

Hill-Climbing

The need for first-order derivative information is an important drawback of gradient descent algorithms. In many application domains such as robotics, the objective function is calculated by running a simulation or a real-world experiment. In this case, it is difficult and sometimes even impossible to derive an analytical formula for computing the gradient. Unfortunately, many real world applications are not even continuously differ-

[1] This feature is due to the fact, that the update rule is only dependent on the sign of the derivative and not its amount. Therefore, the algorithm is not susceptible to noise or error in the estimation of the derivative.

entiable and therefore do not meet the requirements for gradient descent optimization. *Hill-climbing* refers to a family of techniques which are applicable in domains where the gradient is not available, or where other requirements for traditional optimization techniques are not met.

Hill-climbing starts at a random point in the search space. The point is then changed by adding small variations to its coordinates. If some new point has a better objective value, then this point becomes the current point, and the next iteration is started. The algorithm terminates when no further improvement is made. As the name suggests, hill-climbing can be imagined as the process of climbing to the top of the objective function, a local maximum. This is achieved by making random steps and choosing the one going *uphill*. Hill-climbing algorithms belong to the class of *greedy* algorithms, as they always jump to points having a better objective value. In other words, they always strive for uphill movements, even when a downhill movement leads to a better objective value in the long run.

Yuret and De la Maza [Yur94] present a powerful, yet astonishingly simple hill-climbing method called dynamic hill-climbing (DHC). DHC combines several ideas of genetic algorithms and more traditional optimization techniques such as gradient descent. Basically, DHC consists of the following three heuristics:

- Remember the locations of local maxima and restart the optimization program at a place distant from previously located maxima.
- Adjust the size of probing steps to suit the local nature of the terrain (of the objective function), shrinking when probes do poorly and growing when probes do well.
- Keep track of the directions of recent successes, so as to probe preferentially in the direction of most rapid ascent.

Hill-climbing algorithms are easy to implement and can be applied to a wide range of problems. However, they focus on local optimization by design. This means, that they are likely to get stuck in local optima. The simulated annealing [KGV83] algorithm tries to avoid this problem using the analogy of annealing as found in metallurgy. During the process of annealing, a metal is repeatedly heated to a high temperature and then gradually cooled down. This allows the atoms of the material to find low energy crystalline configurations. In contrast to DHC, simulated annealing allows also for downhill movements. For this, a new parameter called temperature is introduced. The temperature is continuously cooled down based on a specified cooling schedule. In each iteration the algorithm generates a random point. If the point improves the objective value, it is accepted. Otherwise, it is accepted with a probability that depends on the temperature. If the temperature is high, even bad points are likely to be accepted. However, when it decreases, points leading uphill are more and more preferred. Other hill-climbing algorithms include, for example, tabu search [GL93], beam search [Bis87], and parallel hill-climbing [Mah95].

Evolutionary Algorithms

Evolutionary algorithms (EA) adopt the concepts and terminology describing biological evolution and apply them to complex optimization problems. Over millions of years, natural evolution has led to the emergence of organisms of extreme perfection and complication. At present, the most accepted theory of evolution is the one formulated by Charles Darwin in his book "On the Origin of Species" [Dar72]. According to the Darwinian theory, evolution favors those species, which are better adapted to their environment. Replication errors during reproduction and other mutations introduce variations into the hereditary information. In turn, these mutations lead to novel characteristics in the affected species. In some cases the novel trait reveals advantageous and improves the organisms chances of survival. In other cases, however, the organism perishes from the population by the effects of such a mutation. This phenomenon is called *natural selection* and can be explained as follows. Organic beings often produce far more offspring than needed. Under optimal living conditions, this leads to an exponential increase of the population size. But finite resources as well as predators act as limiting factors to this growth. The consequence is a fierce competition between the organisms for food, water and other resources. The only rule which characterizes this competition is often termed "survival of the fittest".

EAs act upon a population of solution candidates, the so-called *chromosomes*. As in other optimization algorithms, each solution candidate is a vector of fixed dimensionality. The elements of the vector are called *genes* in reference to the terminology used in genetics. Further, the varying values used at specific positions are called *alleles*. Based on an initial population of chromosomes, EAs simulate natural evolution through the processes of reproduction, mutation, competition and selection. In Algorithm 3 we present a generic evolutionary algorithm.

Algorithm 3 The generic evolutionary algorithm.

Require: $O = (\Omega, f, \succeq)$
1: $t \leftarrow 0$
2: $\mathcal{P} \leftarrow$ create random population of size n, $\mathcal{P} \subset \Omega$
3: `evaluate`$(\mathcal{P}, f, \succ)$ {*calculate the fitness value for each individual*}
4: **while** `not terminate` **do**
5: $\quad\mathcal{P}' \leftarrow$ `select`$(\mathcal{P}, n)$ {*select n parents from population*}
6: $\quad\mathcal{P} \leftarrow$ `crossover`$(\mathcal{P}', m)$ {*create m children through crossover from population*}
7: $\quad\mathcal{P} \leftarrow$ `mutate`$(\mathcal{P})$ {*mutate individuals in population*}
8: $\quad$`evaluate`$(\mathcal{P}, f, \succeq)$ {*calculate the fitness value for each individual*}
9: $\quad t \leftarrow t + 1$
10: **end while**

The population is first initialized with a number of random values (see line 2 of Algorithm 3). Random initialization is mainly performed, because it is not clear in which

part of the search space the optimum lies. If domain specific information is provided, then the start population can be chosen from a particular part of the search space. Once initialization is finished, each chromosome is evaluated using the objective function f. The resulting *fitness* value is used as a measure of the individual's quality with respect to the given problem. The best individuals are selected as parents for the next generation and inserted into the intermediate population P'. Selection eliminates chromosomes which are not appropriate for solving the problem at hand. Good individuals survive, less fit individuals get removed from the population. The parental chromosomes are combined by a crossover operator, in order to form the chromosomes of the descendants. The recombination operator simulates sexual reproduction and defines how an offspring is generated from two parents. Afterwards, the mutation operator introduces random variations into the offspring chromosomes. Altering the individuals in this way ensures genetic diversity and decreases the probability of getting stuck in a local optimum. After application of crossover and mutation the generation counter is increased and the whole process is repeated. This loop is continued until a termination criterion is met.

Typically, an EA terminates after a specified number of generations. However, there are various other termination criteria, such as termination upon convergence, in which the algorithm stops after reaching a desired fitness value. Upon termination, the population member with best fitness is returned as the best solution to the posed problem.

Synthesis of Finger Motions

Using the optimization algorithms and a learned PLDPM, we can generate index finger movements, by taking the position of the button into account. For this, we need to specify the metric of imitation. A simple metric of imitation for 'button-pressing' is based on the distance of the tip to the goal position at the end of a given animation. In the optimal case, the distance between the tip and the goal position is zero. Of course, we can think of more complex metrics, which would include the wrist position or the orientation of the finger. However, for reasons of clarity and comprehensibility we will use a simple metric in the following.

Let Θ be a function, which takes a joint angle configuration as input and returns a corresponding vector of Cartesian coordinates of the fingertip. Further, let $\boldsymbol{b}$ be the position of the button. The metric of imitation (or fitness function) can be defined as:

$$F(\widetilde{\boldsymbol{x}}) = ||\Theta(\Psi^{-1}(\widetilde{\boldsymbol{x}})) - \boldsymbol{b}|| \tag{4.10}$$

Given the above metric of imitation F as objective function, we can use an optimization algorithm to determine the ideal finger configuration. The ideal (two-dimensional) vector $\widetilde{\boldsymbol{q}} \in \mathbb{P}$ for the final frame of the animation, is determined by solving the following optimization problem:

$$\underset{\widetilde{\boldsymbol{q}} \in \mathbb{P}}{\operatorname{argmin}} \quad F(\widetilde{\boldsymbol{q}}) \quad \text{subject to} \quad \Phi(\widetilde{\boldsymbol{q}}) > \tau \tag{4.11}$$

where τ is a threshold for the probability of posture. Postures with a probability below τ are not accepted as solution candidates. In each step of optimization, $\Psi^{-1}(\widetilde{\boldsymbol{q}})$ is computed, and the rotations are applied on a virtual model of the index finger. Then, the discrepancy between the position of the button and the current tip position is calculated to yield the fitness value $F(\widetilde{\boldsymbol{q}})$. The resulting objective value is used by the optimization algorithm to determine new coordinates for $\widetilde{\boldsymbol{q}}$ in the next iteration. This is repeated until the distance between the tip and the button is sufficiently low. Note, that we used $\Phi(\widetilde{\boldsymbol{q}}) > \tau$ as a hard constraint in our optimization. Hard constraints are constraints that must never be violated by potential solutions. However, they can have a negative effect on optimization efficiency because they restrict the exploration process. We will see in later chapters, that we can avoid such problems by including the task constraints in a 'soft' way into optimization.

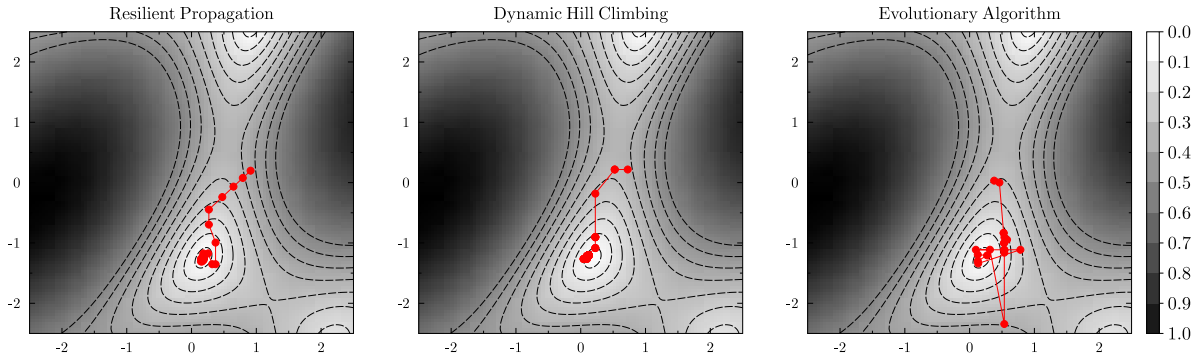


Figure 4.9: Search behavior of different optimization techniques. The red points show the solution candidates that are evaluated during the optimization process.

Figure 4.9 depicts this search process for three different optimization techniques. The colormap and the corresponding contour plot represent the objective value at different parts of the search space. White spots represent high-fitness areas of the search space, while dark areas correspond to low-fitness areas. The red trajectory shows the position of $\widetilde{\boldsymbol{q}}$ after each iteration of the optimization algorithm. We can see in Figure 4.9 (left) how RPROP slowly converges towards the white high-fitness area of the search space. It does so by empirically estimating the gradient and then searching in that direction. If we compare it to DHC (Figure 4.9 (middle)), we see that the search strategy of DHC gets faster to the high-fitness area. After each successful move DHC multiplies the stepsize by two. As a result, it makes larger steps towards the optimum. In contrast to that, RPROP has a nearly equidistant stepsize between successive iterations on this objective function. A totally different search strategy is observed for the EA (Figure 4.9 (right)). It performs large, chaotic looking jumps in the search space. Thus, improvement is not steady but rather abrupt. This can be advantageous when exploring large search spaces. On the other hand the algorithm might overlook local optima because of this strategy.

	Optimization Error		
	RPROP	DHC	EA
Average	0.0257	0.0256	0.0697
Std. dev.	0.0257	0.0172	0.0441

Table 4.2: Averaged optimization error over 100 trials of the 'button-pressing' imitation task for different optimization algorithms.

In Table 4.2 we see the distance between the finger tip and the button position averaged over 100 trials. In each trial the position of the button was changed and the distance between tip and button after posture synthesis was calculated. For a fair comparison, in each trial, optimization was run for exactly 400 evaluations of the objective function. The dimensionality reduction technique used in all experiments was PCA. The two best performing optimization algorithms for the 'button-pressing' task are the DHC and RPROP algorithm. DHC was even slightly better than the RPROP algorithm, with a significantly lower standard deviation of the results. The results of EA in this problem domain are not satisfying. In later examples found in this thesis, we will see that evolutionary search becomes preferable when faced with more complex search spaces containing many equally good local optima.

	Optimization Error						
	PCA	LLE	CCA	CDA	MDS	ISOMAP	NLM
Average	0.0256	0.0417	0.2530	0.3535	0.0577	0.1456	0.0631
Std. Dev.	0.0173	0.0559	0.2672	0.2446	0.0656	0.1169	0.0782

Table 4.3: Optimization error on the 'button-pressing' task, when using different dimensionality reduction techniques for learning of the PLDPM.

In the next experiment, the DR techniques were varied, in order to understand the influence of DR on posture synthesis. Table 4.3 summarizes these results. Here, the best performing algorithm found in the earlier experiment, namely DHC, was used in conjunction with PLDPMs learned using different DR techniques. Apart from the DR method, the execution of the experiment remained the same. The results achieved using PCA outperform all other methods sometimes by a factor of ten. The reason for this phenomenon can be understood by analyzing the structure of the objective function given different DR techniques, as shown in a 3D plot in Figure 4.10. In contrast to Figure 4.9, here, the fitness value of each point is represented by the height of the

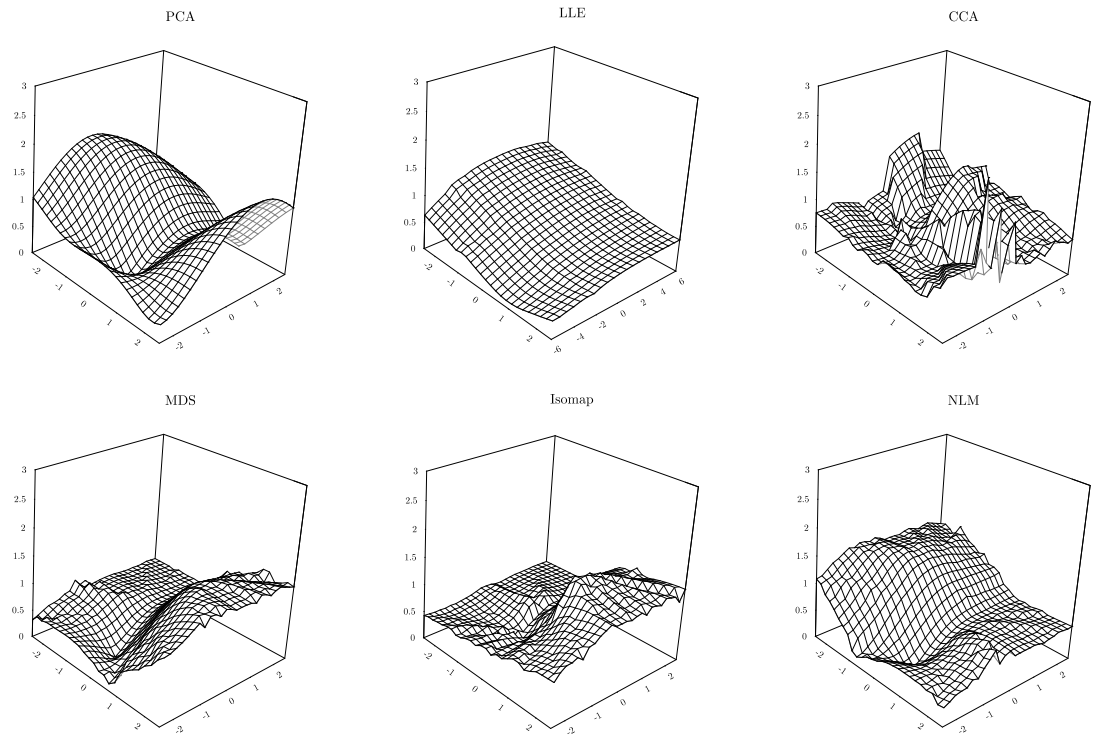


Figure 4.10: Fitness landscapes for different dimensionality reduction techniques.

landscape at that point. The result is a landscape that henceforth will be called *fitness landscape*. As can be seen, the fitness landscape for PCA is a smooth, differentiable function with one local minimum. Optimization on such a smooth landscape can be better understood by imagining the behavior of a virtual marble on top of it. In the case of PCA, it does not matter where we place the marble; because of the smooth landscape, the marble will always travel downhill, until it settles down at the local minimum. The local minimum is said to act as an *attractor*. The fitness landscape for LLE also exhibits a smooth surface, which is also reflected in a low optimization error achieved. A slightly higher optimization error is produced by NLM and MDS, which have rugged but still continuous landscapes. In the case of CCA, however, the fitness landscape is highly rugged and contains many local optima. The optimization error is significantly higher than with the methods analyzed before. Using the marble analogy we could say: It is unclear where the virtual marble will settle at the end of the optimization process. This phenomenon is a sideffect of the strong nonlinearities introduced by the DR and renders posture synthesis more difficult. This clearly shows that the fitness function is dependent upon the DR technique employed. Ideally, the combination of DR and fitness function should lead to a smooth, unimodal (only one optimum) fitness function. This property is often fulfilled when using PCA as a DR technique.

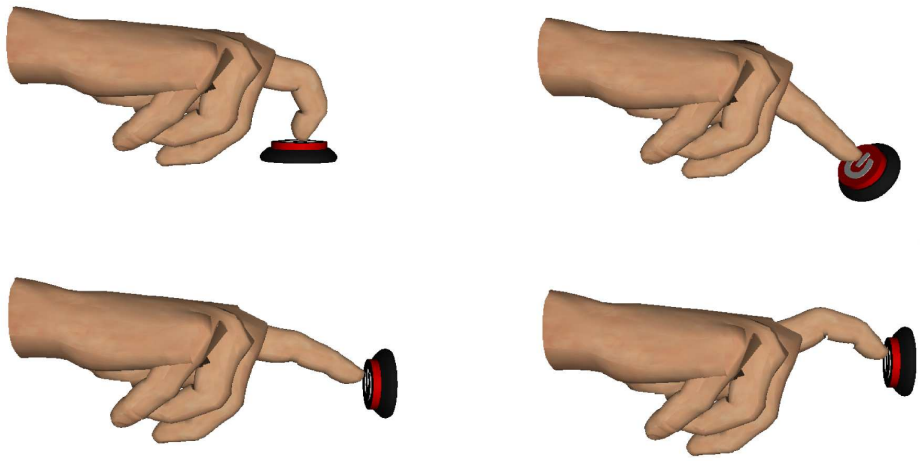


Figure 4.11: Synthesized final postures of 'button-pressing' behavior. The shape of the virtual hand depends on the postion of the button.

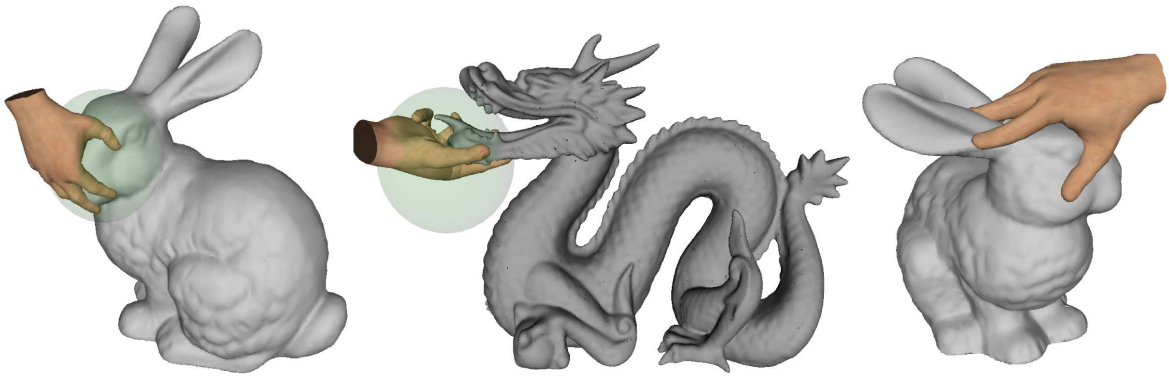
Figure 4.11 depicts a set of hand configurations optimized for different locations of the button. We only optimized the final hand configuration during the 'button-pressing' task. Hence, the depicted pictures only show the final hand configuration. To produce an animation of the full movement of the hand, all that needs to be done, is to reproject the points along the trajectory going from the starting point, through the preshape point to the (optimized) final point in the posture space.

4.5. Conclusion

In this chapter, we presented a general approach for learning motor behaviors by imitation. The approach consists of three steps and is neither restricted to a particular synthetic humanoid, nor to a specific motion recording technology. Further, we introduced the probabilistic low-dimensional posture model construction algorithm. This algorithm exploits the fact that natural, human motion can often be described as a combination of a small number of components. The result of the algorithm are compact representations of observed human motion and the inherent anatomical constraints. In conjunction with optimization techniques, these representations can be used to imitate the observed behavior. The imitation and motion synthesis process adapts the behavior to the current context, e.g. the current position of the button.

While for explanatory purposes we used a simple application example in this chapter, we will show in the next chapter that the approach can also be used to learn more complex motor abilities. In particular, we will show how natural human grasping can be imitated using this approach.

5. Learning to Imitate Natural Human Grasping



5.1. Introduction

Grasping is considered as one of the most flexible and complex abilities of human beings. We use it in our everyday life to lift, transport or manipulate an object. The importance of grasping is not limited to physical interaction with our environment. There is also scientific evidence suggesting that the emergence of complex grasping abilities is an important step in the evolution of human language [Arb08].

Providing synthetic humanoids with grasping abilities is also regarded as one of the great challenges of robotics and computer animation. With the advent of highly realistic android robots and computer animated actors, the automatic generation of grasping behavior has become a key issue in these fields. In order to act in a convincing way in a virtual environment, virtual humans need to be able to perform grasping tasks on a variety of everyday objects. This calls for fast and efficient *grasp synthesis* algorithms that produce high quality hand shapes with natural appearance. However, this remains a challenging task, as evident from the following quote:

> "the human hand is a highly complex structure that in many ways defies understanding" [MI94]

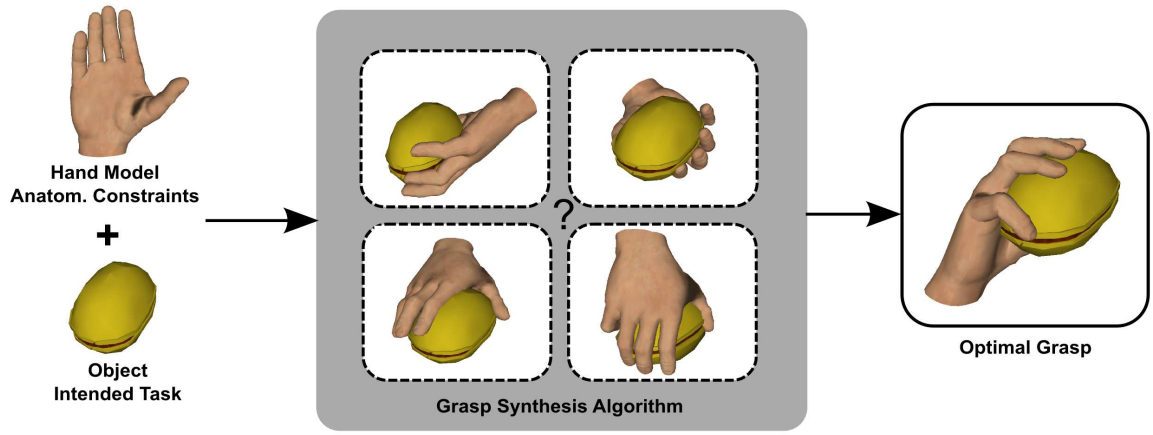


Figure 5.1: The general flow of a grasp synthesis algorithm. A model of the used hand as well as a set of constraints is supplied, along with information on the task intended after grasping. An optimization algorithm evaluates a large number of potential solutions and searches for a hand shape which best fits the specified constraints.

A high number of interdependent degrees of freedom in the hand makes the generation of grasping behaviors particularly difficult. At the same time, the contact of the hand with the object surface must be optimized to ensure the stability of the grasp. Pure motion capture, as used in computer animation, does not solve the problem. More specifically, motion capture does not provide support for the task of closely fitting the hand shape to new objects, i.e. the retargeting problem [Gle98].

In the following a PLDPM based method for the efficient synthesis of natural looking hand shapes for grasping animations is presented. A first step is to collect a set of hand shapes using motion capture techniques. Then, the PLDPM approach is used to create low-dimensional models of these postures. It was already reported by Santello and colleagues that the first two and three principal components account for more than 80% and resp. 87% of the variance in hand posture [SFS98]. The space encoded in the learned model is small enough to be searched through at interactive rates using optimization strategies. Additionally, because the grasp is synthesized from a PLDPM, the resulting hand configurations imitate the human grasps supplied during learning.

5.1.1. Problem Statement

In the following, we will consider the problem of generating natural grasps for synthetic humanoids. Given a geometric description of a target object, our goal is to generate a hand shape that results in a natural-looking and stable grasp. The process can be seen in Figure 5.1. In order to generate such a hand shape, an algorithm needs to take the geometry of the object, as well as its position and orientation into account. However, for most objects, e.g. a bottle, there exist many possible ways to grasp them.

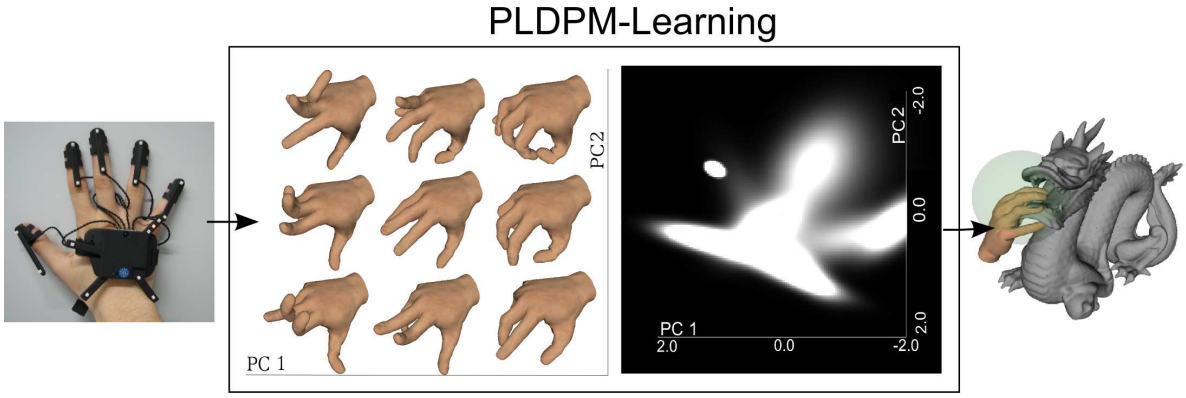


Figure 5.2: The proposed grasp synthesis method: recorded motion capture data (left) is transformed into a probabilistic low-dimensional posture space (middle). Using optimization techniques the space can then be searched for appropriate grasps of new objects.

Therefore, a task description is needed, which reduces the redundancies and ensures that the generated grasp best suits the task at hand. Still, for many objects there is no single 'optimal' way of grasping. Following [Pol94], the problem of grasp synthesis can be stated as follows. Given:

- a geometric model of the target
- a specification of task
- a geometric and kinematic model of the synthetic hand

find a grasp for the target object that is suitable for the given task. One way to achieve this goal is to write specialized programs for each grasping task. For example, we can write a program that grasps a bottle. Given the sensory information about the location and orientation, the program generates a hand posture that grasps the bottle. Such approaches require expert knowledge and are typically restricted to a set of predefined object shapes. Additionally, such approaches do not ensure that the resulting grasp looks convincingly human-like. While this requirement is often not essential in robotics, especially when anthropomorphism is not intended, it is a fundamental necessity in most computer animation scenarios.

In the following we will present a different approach to the problem. Based on the imitation learning technique introduced in Chapter 4, the grasp synthesis algorithm consists of three steps: observation, representation and reproduction. This can be seen in Figure 5.2. During the observation phase a set of example grasps is recorded using a fingertracking device. After that, the recorded data is used to learn one or several PLDPMs for grasping. As can be seen in the figure, some grasps within the posture space are anatomically infeasible. The probability density function (grayscale graph in Figure 5.2) encoded by the GMM helps to identify these grasps. Using optimization

techniques we can then search for anatomically feasible grasps which best fit a given target object and task.

The hand shapes generated by this algorithm closely resemble the recorded examples. This feature can be exploited to synthesize particular types or styles of grasps. For example, if we train the PLDPM with spherical grasps solely, then the generated hand shapes will always be spherical. Another feature of this algorithm is that not only the hand shape, but also additional parameters such as the hand position and orientation are optimized.

In the following we will introduce the details of this grasp synthesis algorithm. First, we discuss the modeling and representation of human grasps within a computational setting in Section 5.2. In Section 5.3, we focus on the metric of imitation. We introduce measures for assessing the quality of a given grasp. In Section 5.4 we then explain the optimization algorithm, as well as possible initialization strategies and computational speedups. Finally, in Section 5.4 we evaluate the proposed algorithm on a set of virtual objects and discuss the results.

5.2. Modeling the Human Hand

This section explains how the human hand and grasping in general is modeled with the PLDPM approach. This includes descriptions of the kinematic model, grasp taxonomy and hand sensors. The concepts and terminology introduced here define the foundation for our grasp synthesis algorithm.

5.2.1. Grasp Taxonomies

The human hand is capable of taking on a great variety of shapes. Despite that, most of the time we employ only a small set of re-occurring hand shapes. Whether we grasp an orange, an apple or a tennis ball the resulting hand shape is similar. Since the beginning of the 20th century, scientists have been trying to categorize human hand configurations into a discrete set of equivalence classes. Such a *taxonomy* facilitates the analysis and documentation of hand shapes as well as their duplication by mechanical devices.

One of the first and most widely used taxonomies for human grasps was put forth by Schlesinger [Sch19] (see Figure 5.3). After the end of the first World War there was a substantial need for an analytic and systematic approach to the design of prosthetic hands. Schlesinger analyzed the functionality needed to perform different grasping and manipulation tasks and proposed a suitable taxonomy. Building on this work, Taylor and Schwarz [TS55] introduced English names for the most important grasps investigated by Schlesinger: cylindrical, tip, hook, palmar, spherical and lateral grip. According to [MI94], the Schlesinger grasps have the following characteristics:

- **Spherical:** A hand shape for grasping spherical objects that can be enveloped by the human hand. The fingers spread, while the palm is arched. The thumb

Figure 5.3: The grasp types of the Schlesinger taxonomy.

stabilizes the object in the hand by exerting opposing forces in direction of the other fingers.

- **Cylindrical:** A hand shape for grasping cylindrical objects such as a cup, a beer mug or a hammer. The four fingers envelop the object and press it in direction of the palm.
- **Hook:** A hand shape suitable for grasping objects with a handle, e.g. a suitcase. The handle presses against the palm, while the hand takes the form of a hook.
- **Lateral:** A hand shape for grasping small, flat objects, e.g. disks, keys, or a piece of paper. The object is held between the curled index finger and the pulp of the thumb.
- **Palmar:** A hand shape for grasping flat, thick objects, e.g. books, videotapes, or a matchbox. The object is held by opposing forces exerted by the thumb on one side, and the index finger and potentially other fingers on the other side.
- **Tip:** A hand shape suitbale for grasping small objects, e.g. a needle, a pen or a marble. The object is held between the tip of the index finger and the tip of the thumb.

The elegance of the Schlesinger taxonomy lies in the fact that it accounts for a large number of situations using only a small number of distinct grasp types. Other taxonomies have, for example, been proposed by Napier [Nap56], Cutkosky [Cut89], and Kamakura [KMI+80]. The Napier taxonomy differentiates between only two types of grasps: the power grip which clamps an object firmly using the palm, and the precision grip where the thumb and other fingers pinch the object. Obviously, the distinction

between only two types of grasps is too coarse to allow for the degree of control that we are aiming for. The taxonomies by Cutkosky and Kamakura have 16 and 14 grasp types, respectively. However, many of the grasp types are very similar to each other and differ only in the task to be executed afterwards. In the remainder of this chapter we will therefore focus on the Schlesinger taxonomy.

5.2.2. The Kinematic Model

In order to generate realistic hand shapes in virtual environments, we need a sufficiently realistic model of the human hand. In particular, we are interested in the hierarchy of bones and joints used for articulation. Figure 5.4 (left) shows the anatomy of the human hand, including all bones and joints.

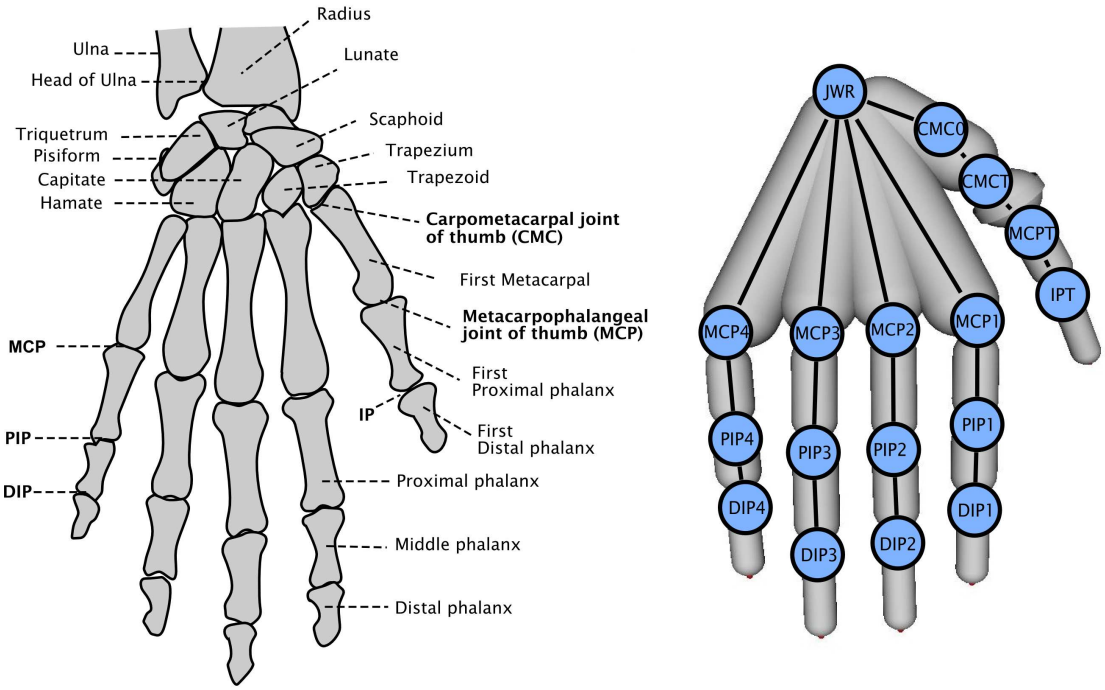


Figure 5.4: Left: The anatomy of the human hand. Right: A simulated skeletal representation of the human hand.

The proximal interphalanageal joints (PIP) and the distal interphalangeal joints (DIP) of the fingers (introduced in Section 4.4), as well as the interphalangeal joint (IP) of the thumb, are so-called *hinge* joints. These joints are only capable of flexion or extension, and have therefore only one degree of freedom. In contrast, the metacarpophalangeal joints (MCP) have a second DOF for adduction and abduction. The carpometacarpal joint (CMC) of the thumb is said to have a third DOF, because the two axes of rotation are not completely perpendicular to each other.

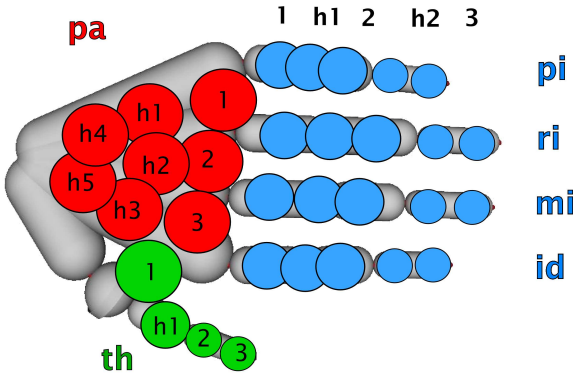


Figure 5.5: Size and position of the sensors in the virtual hand model.

In our approach, the hierarchy of bones is modeled using skeletal animation techniques. The virtual skeleton consists of virtual bones arranged in a kinematic chain. The topology of the kinematic chain is based on the human anatomy. Each element in the kinematic chain has a local coordinate system which is dependent on all anterior coordinate systems in the hierarchy. Figure 5.4 (right) shows a virtual model representing the skeleton of a simulated human hand. In contrast to the earlier analysis of the biological human hand, the virtual skeleton does not include any limitation of possible finger rotations so far. Each joint is therefore modeled as a two DOF joint. In order to account for the third DOF of the thumb, an additional joint is added here. Additionally, the model also includes a joint for the wrist.

Given the above virtual skeleton, a variety of shapes of the human hand can be generated. This is done by applying a rotation on each of the modeled joints. After applying the rotations, the hierarchy of the skeleton is traversed and the new world coordinates of the virtual bones are computed and applied to the rigid bodies representing the bones. Note that the system does not include any model on the anatomical constraints of the hand, so far. Generated grasps may not necessarily appear human-like. We will see later, how this problem can be overcome by utilizing statistical information gathered from example grasps.

5.2.3. The Sensor Model

Sensor models are often used for proximity computation in virtual grasping [ST94]. Concretely, they are used for fast collision detection between the simulated hand and scene objects or for computing distances. Using such sensors enables us to perform these computations without having to resort to the complete geometry of the hand. Based on this idea, we fit a set of spherical sensors to the introduced hand model.

Figure 5.5 shows the size and position of the sensors in the virtual hand. The sensors have different radii depending on their position and task:

- **Large:** Four sensors at the end of the metacarpal bones in the palm. ($pa1, pa2, p3$ and $th1$). The radius of the sensors is 1.2 cm.
- **Normal:** Nine sensors located at the joints of the fingers. ($th2, id1, id2, mi1, mi2, ri1, ri2, pi1$ and $pi2$). The radius of the sensors is 0.9 cm.
- **Small:** Five sensors located at the tip of the fingers. ($th3, id3, id3, mi3, ri3$ and $pi3$). The radius of the sensors is 0.7 cm.

The area of the palm and the fingers is filled with additional sensors $pah1$ to $pah5$ and in the empty spaces within the fingers the sensors $thh2, idh1, idh1, idh2, mih1, mih2, rih1, rih2, pih1, pih2$ are added. These sensors are called *helpers* as they are only used to ensure that the space between the major sensors is free of any collision with an object.

Not all sensors are used when a grasp is optimized. Depending on the grasp type, a set of relevant sensors is selected. Additionally, the sensors are categorized depending on their importance. More specifically, the sensors are classified into three distinct groups. Primary sensors $\mathbb{S}_P$ are directly involved in the formation of the grasp. They strictly need to touch the surface of the target object. Secondary sensors $\mathbb{S}_S$ are only indirectly involved in the grasp. Secondary sensors need to be as close as possible to the surface of the target object. Finally, helper sensors $\mathbb{S}_H$ are not involved in the grasp at all. They only need to ensure that no collision with the object occurs.

Table 5.1 summarizes the assignment of primary, secondary and helper sensors based on the grasp type to be performed. In our earlier introduction of the Schlesinger grasp types we specified that for a *tip* grasp only the tips of two fingers exert opposing forces on the object. This fact is reflected in Table 5.1. The tip grasp requires only two sensors to be touching the object, namely the $th3$ and $id3$ sensors.

5.2.4. Grasp Parametrization

We will now address the question of how to parametrize a grasp such as to support later optimization. Our goal is to find a minimal set of parameters that are sufficient to uniquely describe a large number of grasps. This set should be minimal in order to ensure that optimization can be efficiently performed. We will divide the parameter set into *extrinsic* and *intrinsic* parameters. Intrinsic parameters specify the shape of the hand, while extrinsic parameters specify its position and orientation.

As intrinsic parameters we will use the low-dimensional coordinates of the grasp. In mathematical terms, a point $\widetilde{\boldsymbol{q}}$ in posture space $\mathbb{P}$ fully specifies the intrinsic parameters of the corresponding grasp. By reprojecting the point $\widetilde{\boldsymbol{q}}$ back into the original space of joint rotations, we get all information necessary to realize the hand posture. We will

Grasp Type	Primary Sensors	Secondary Sensors	Helper Sensors
Spherical	$th3, mi3, pi3$	$pa2, id3, ri3$	$pa1, pa3, th1, th2,$ $id1, id2, mi1, mi2,$ $ri1, ri2, pi1, pi2$
Cylindrical	$pa1, pa2, pa3, th1$	$th3, id3, mi3, ri3, pi3$	$th2, id1, id2, mi1,$ $mi2, ri1, ri2, pi1,$ $pi2$
Hook	$pa1, pa2, pa3, pi1$ $id1, mi1, ri1,$	$id3, mi3, ri3, pi3$	$th1, th2, th3, id2,$ $mi2, ri2, pi2$
Palmar	$th3, id2, id3$	$th1, th2, id1$	$pa1, pa2, pa3, mi2,$ $mi3, ri1, ri2, ri3,$ $pi1, pi2, pi3$
Lateral	$th3, id2$	$id1, id3$	$th1, th2$
Tip	$th3, id3$	none	$th1, th2, id1, id2$

Table 5.1: Categorization of sensors into primary, secondary and helper ssensors based on the grasp type to be performed.

introduce a local coordinate system, the grasp coordinate system (GCS), to specify the extrinsic parameters. To define the GCS we need to set the origin $\boldsymbol{b}_o$ and the two axes $\boldsymbol{b}_x$ and $\boldsymbol{b}_y$ of the coordinate system. The third axis $\boldsymbol{b}_z$ can be trivially computed by $\boldsymbol{b}_z = \boldsymbol{b}_x \times \boldsymbol{b}_y$. The position of the hand is always specified within the GCS and must be transformed into global coordinates before application.

Similarly, the orientation of the wrist is locally specified in the GCS by $\boldsymbol{\alpha} \in \mathbb{R}^3$. Each of the entries in $\boldsymbol{\alpha}$ specifies a rotation angle for rotation around one of the axes of the GCS. To summarize, an arbitrary grasp can be uniquely specified by setting the parameters:

- **Position** of the wrist $\boldsymbol{p} \in \mathbb{R}^3$
- **Orientation** of the wrist $\boldsymbol{\alpha} \in \mathbb{R}^3$
- **Shape** of the hand in a posture space $\widetilde{\boldsymbol{q}} \in \mathbb{P}$

Figure 5.6 visualizes the specification and use of a GCS. Typically, the GCS is specified relative to the object to be grasped, e.g. on the surface of the ball. The orientation of the GCS can be set based on the task to be achieved or the direction from which the object needs to be grasped. Initializing the hand position inside the GCS ensures that the extrinsic parameters take on reasonable values. The hand will be initialized near the object and it will be correctly oriented.

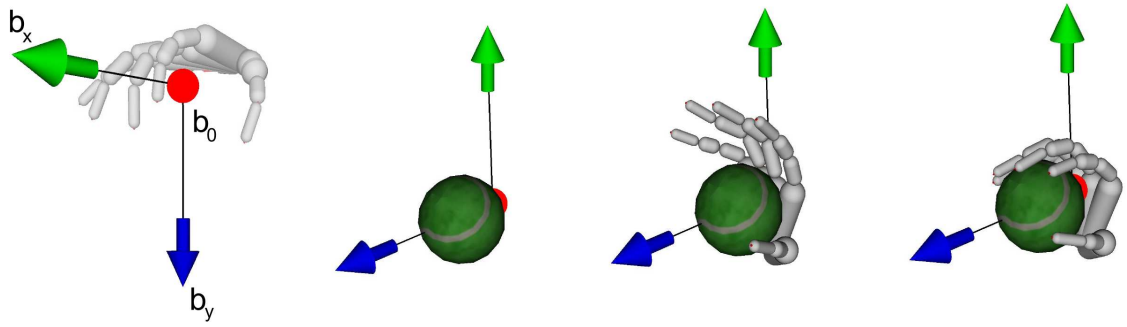


Figure 5.6: The grasp coordinate system. The GCS specifies the extrinsic parameters of the hand, i.e. the position and orientation of the hand. These values can be determined based on the object to be grasped, e.g. the Tennis ball.

5.3. Grasp Quality Measure

As discussed in Chapter 4, a PLDPM requires the specification of an imitation metric which defines a measure of how successful a given attempt at imitation is. In the case of grasping, the metric of imitation needs to evaluate and rate the quality of a given hand shape. Metrics for grasping have long been the focus of studies in robotics and can be summarized by the term *grasp quality measures* [MF96].

Given the geometric (and physical) properties of an object, as well as the grasp to be performed, a grasp quality measure computes a scalar fitness value which measures the suitability of the intended grasp with respect to the object. For the generation of natural and successfull grasps, we need a reliable grasp quality measure that combines various important aspects, e.g. anatomy and stability, when computing the fitness values.

Many grasp quality measures in the literature focus on the physical properties of a grasp. The goal is to produce stable grasps, meaning that the grasp resists the largest possible set of external forces or torques. Following this rationale, many grasp quality measures are based on the concept of wrench spaces. Wrench spaces are used to determine the set of all external disturbances that can be balanced by a particular grasp configuration. Other grasp quality measures use the geometric relationships between fingers and the object in order to assess the quality of a grip.

The following section introduces a simple grasp quality measure which targets applications in computer animation and virtual reality domains. The quality of the grasp is computed as a sum of three components:

- The **stability** F_S of the grasp
- The **distance** F_D of the fingers to the object
- The **anatomical plausibility** F_A of the hand shape

The overall grasp quality measure is therefore a function F:

$$F(\boldsymbol{p}, \boldsymbol{\alpha}, \widetilde{\boldsymbol{q}}) = F_D(\boldsymbol{p}, \boldsymbol{\alpha}, \widetilde{\boldsymbol{q}}) + F_S(\boldsymbol{p}, \boldsymbol{\alpha}, \widetilde{\boldsymbol{q}}) + F_A(\widetilde{\boldsymbol{q}}) \tag{5.1}$$

In contrast to other metrics, this measure mainly focuses on the natural appearance and visual quality of the generated grasps. The main goal of this measure is to evaluate how convincing the appearance of a grasp is. While basic stability properties are taken into account in this evaluation, other physical aspects of the generated grasp, such as force closure or grasp wrenches, are not considered. The main reason for this choice is efficiency. The targeted application domains often do not rely on physical models and are mainly concerned with the visual quality of the results. Yet, the algorithm which we will present works with any arbitrary grasp quality measure. Other metrics, like the ones used in the robotics literature can be employed, too.

Before we discuss the components of this grasp quality measure in more detail, we will first introduce the following violation function $V_B(m)$. We will use the violation function in order to introduce a nonlinear scaling for the penalties computed for each of the components of the fitness function. The violation function has the following general form:

$$V_B(m) = \frac{\mathrm{e}^{-2}}{1 - \mathrm{e}^{-2}} \cdot (e^{2 \cdot \frac{|m|}{B}} - 1) \tag{5.2}$$

The above function has exponential growth and is symmetric with respect to the ordinate. Further, the equation $V_B(\pm B) = 1$ holds. Hence, the parameter B can be used to control the degree of exponential growth of the function. The violation function is used to weight the contribution of different components to the overall fitness function. The function takes an error value computed by a grasp metric as input. The violation function then scales this value nonlinearily. As a consequence, high error values are more severly penalized in comparison to small error values. The specific values for B and m are, however, dependent on the actual component of the fitness function and will be discussed in more detail in the following section.

5.3.1. Finger Distance

A Schlesinger grasp requires touching the object with at least two fingers. The fingers involved in the grasp must have contact with the object's surface. In contrast to a real situation, in simulated environments the fingers can also penetrate the object. However, to synthesize a natural grasp, we need to ensure that no such finger–object penetration occurs.

The F_D component of the grasp quality measure uses the distances between the finger sensors and the object surface to realize this goal. In an ideal grasp, all primary and secondary sensors touch the surface of the object. This is the case if for every sensor s the distance d_s between the sensor and the object surface is zero. If this is not the

case, then one of the following two situations apply: (i) If d_s is negative, then the sensor is inside an object, i.e. a penetration occurred. (ii) If the distance is positive, then the sensor is on the outside the object and is not in contact with it.

The finger distance penalty term F_D is computed as a sum of individual distance penalties computed for each of the finger sensors s. The individual distance penalties are based on the particular type of the analyzed sensor. For example, helper sensors are only used to make sure that no penetration between the object and other parts of the hand occurs, i.e. they only contribute to the distance term, if d_s is negative. Therefore, the overall finger distance penalty term can be calculated as follows:

$$F_D = \sum_{s \in \mathbb{S}} \begin{cases} V_{0.5}(d_s) & \text{if } s \text{ is a primary sensor} \\ V_{1.0}(d_s) & \text{if } s \text{ is a secondary sensor} \\ V_{0.3}(d_s) & \text{if } s \text{ is a helper sensor} \end{cases} \tag{5.3}$$

F_D computes a weighted sum of the distances between the sensors and the object surface. If the sensor is a primary sensor, then a distance of 0.5 cm is sufficient in order to generate a penalty value of one. If s is a secondary sensor, then a distance of 1 cm is needed to produce the same penalty as for a primary sensor. Helper sensors only produce penalties if they are located inside the object. However, such a penetration is highly penalized, and therefore the function V will produce large values in this case.

5.3.2. Anatomical Plausibility

The goal of the grasp optimization process is to find a natural looking hand shape leading to a stable grasp on a user provided 3D object. An important question in this regard, is how to ensure that the generated grasp is natural, or in other words, anatomically plausible. Instead of relying on biomechanical models, we will use the probability density function of the PLDPM for solving this problem.

The probability $\Phi(\widetilde{\boldsymbol{q}})$ of a grasp estimates the likelihood of the vector of the posture space $\widetilde{\boldsymbol{q}}$ obeying the same statistical model as the recorded training grasps. Assuming that enough grasps were used for training, this result is an indicator for the anatomical feasibility of the hand shape. Accordingly, the penalty function for the anatomical plausibility of a grasp can be defined as follows:

$$F_A(\widetilde{\boldsymbol{q}}) = \frac{\epsilon}{1-\epsilon} \cdot \left(\frac{\Phi^{max}}{\Phi(\widetilde{\boldsymbol{q}})} - 1 \right) \tag{5.4}$$

where Φ^{max} is the maximal value of the probability density function. The constant ϵ is used to normalize this function. It specifies the percentage of the maximal value Φ^{max} which is still sufficient to regard a given grasp as natural. The penalty is zero, if the probability of the grasp is maximal, i.e. if the grasp is perfectly natural. On the other hand, the penalty is one, if $\Phi(\widetilde{\boldsymbol{q}}) = \epsilon \cdot \Phi^{max}$, i.e. for unrealistic grasps. In all of the experiments of this thesis, we use the value $\epsilon = 0.01$.

5.3.3. Stability

Another important aspect that needs to be included in the evaluation of potential hand shapes for grasping is stability. Analyzing the stability of a given grasp, we can decide if the object is correctly fixated for later manipulation tasks. According to [RS06], the quality of a grasp with respect to stability can be determined by analyzing the following features:

- The location of contact points on the object
- The configuration of the finger joints
- A combination of above two points

We will use a stability estimation measure which is based on the contact points between the finger sensors and the object. However, during optimization many potential grasps will be generated, for which the sensors are not located on the object surface. To circumvent this problem, we will first compute the nearest positions on the object surface. These are taken as estimates of the contact points of the sensors with the object. Next, the stability of the resulting set of contact points is estimated using a number of heuristics, which are derived from literature on robotic and human grasping. For grasp types that produce enveloping grasps, such as the spherical grasp, the stability grasp index (SGI) [RS06] will be used for estimating the corresponding stability. For grasp types that are based on oppositions of finger groups, such as the palmar grasp, a combination of the cone of friction (COF) [MI94] and a novel contact normal opposition (CNO) measure will be employed.

In the case of the SGI, the stability is measured by analyzing the inner angles of a planar polygon spanned between the contact points. This is motivated by the fact that contact points are ideally uniformly distributed on the object surface. To quantify the distribution of the fingers on the object, we can calculate how much the internal angles of the grasp polygon deviate from those of a regular polygon. In the following we will always decompose the grasp polygon into a set of triangles and compute the SGI on the resulting set. Figure 5.7 shows an example for the computation of the grasp stability index. The inner angles of two triangles spanned between the contact points of different sensors with the object are evaluated. According to [RS06], given a triangle with internal angles $\boldsymbol{\beta} \in \mathbb{R}^3$, the SGI can be computed as:

$$M_{sgi}(\boldsymbol{\beta}) = \frac{1}{240} \cdot \sum_{i=1}^{3} |\beta_i - 60| \tag{5.5}$$

In the worst case, the triangle degenerates into a line. In this case, one internal angle is 180° while the remaining two internal angles are zero. Consequently, the SGI of such

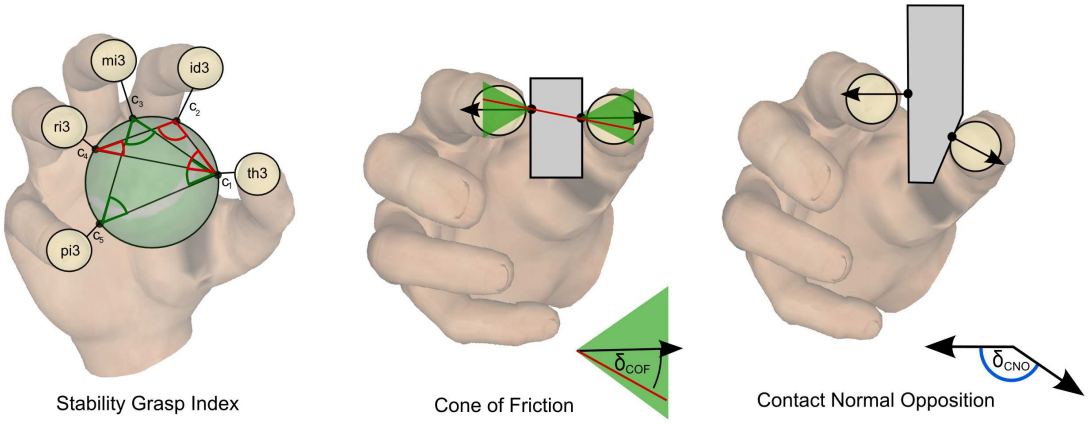


Figure 5.7: The three quality measures used for grasp evaluation. Left: The grasp stability index analyzes the inner angles of a planar polygon spanned between the contact points. Middle: The cone of friction measure computes the angle between the contact normal and the line connecting the contacts. Right: The contact normal opposition measure computes the angle between the contact normals.

a configuration will be one, as exemplified by the following equation:

$$1 = \frac{1}{240} \cdot \Big(\, |180° - 60°| + |60° - 0°| + |60° - 0°| \, \Big) \tag{5.6}$$

For an equilateral triangle (the best case), the SGI will be zero. The result of the SGI is then used as input to the violation function in order to compute the overall penalty:

$$F_{sgi}(\boldsymbol{\beta}) = V_{0.4}\Big(M_{sgi}(\boldsymbol{\beta})\Big) \tag{5.7}$$

The set of triangles that are evaluated to compute the SGI is dependent on the choice of the grasp type. We will discuss this aspect in more detail below.

Another metric for measuring the stability of a grasp is the so-called cone of friction. According to MacKenzie and Iberall [MI94], the cone of friction is a geometric interpretation of the maximally allowed angle ω between the surface normal and the applied force vector. If the applied force at the contact point makes an angle δ_{cof} with $|\delta_{cof}| < \omega$, then no slip will be produced at the fingertip. The angle ω is determined by the coefficient of friction of the grasped object. To determine if a grasp induced by two antagonist finger sensors is stable, we first compute the vector $\boldsymbol{c}$ connecting the corresponding contact points. The angle δ_{cof} can then be calculated using:

$$\delta_{cof} = \arccos\left(\frac{-\boldsymbol{c} \cdot \boldsymbol{n}}{||\boldsymbol{c}|| \cdot ||\boldsymbol{n}||}\right) \tag{5.8}$$

where $\boldsymbol{n}$ is the surface normal at the contact point. In the ideal case, the angle δ_{cof} equals zero meaning that the vector of the applied force is parallel to the surface normal. In the same manner as for SGI, the angle δ_{cof} is weighted using the violation function. Hence, the penalties according to the COF criterion can be computed by the following formula:

$$F_{COF} = V_{45}(\delta_{cof}) \tag{5.9}$$

In Figure 5.7 (middle) the cones of friction are depicted as green triangles centered at the contact points. The angle δ_{cof} is spanned between the contact normal (black arrow) and vector $\boldsymbol{c}$ (red line).

Finally, as a last criterion for estimating the stability of a grasp, we introduce the contact normal opposition (CNO) measure. The CNO measure evaluates the angle between the two surfaces where the investigated antagonistic sensors make contact with the object, as illustrated in Figure 5.7 (left). Let $\boldsymbol{n}(1)$ and $\boldsymbol{n}(2)$ be the two contact normals, then the angle δ_{cno} can be computed as follows:

$$\delta_{cno} = \arccos\left[\frac{\boldsymbol{n}(1) \cdot \boldsymbol{n}(2)}{||\boldsymbol{n}(1)|| \cdot ||\boldsymbol{n}(2)||}\right] \tag{5.10}$$

In the ideal case, the two contact surfaces are parallel, and the contact normals face opposite directions. In the worst case, the contact normals both point in the same direction. After computing the values for δ_{cno}, they are normalized according to the general scheme leading to the penalty function:

$$F_{CNO} = V_{23.5}(\delta_{cno}) \tag{5.11}$$

Consequently, the function returns a penalty of one, if the angle δ_{cno} equals 23.5°. The stability metrics introduced are used in different combinations for estimating the stability of a given grasp. Note, for example, that the CNO measure alone is generally not practical as a grasp stability measure and will subsequently always be used in conjunction with the COF. Next, we will discuss the stability analysis process with respect to the intended grasp type.

Enveloping Grasps

We will use the term enveloping grasps to refer to spherical, cylindrical or hook grasps. In all of these grasps the fingers are flexed, so as to envelop the target object. For all enveloping grasps the SGI is used to estimate the stability. The only difference between these computations are the sensors used.

In the following we will use the $\blacktriangle(s_1, s_2, s_3)$ operator to specify a polygon (triangle) with vertices located at the position of the sensors s_1, s_2, and s_3. For the derivation of the stability we make use of the following polygons:

Polygon	**Angles**
$\blacktriangle_1(th3, id3, ri3)$	$\boldsymbol{\alpha}(1) = (\boldsymbol{\alpha}_{11}, \boldsymbol{\alpha}_{12}, \boldsymbol{\alpha}_{13})$
$\blacktriangle_2(th3, mi3, pi3)$	$\boldsymbol{\alpha}(2) = (\boldsymbol{\alpha}_{21}, \boldsymbol{\alpha}_{22}, \boldsymbol{\alpha}_{23})$
$\blacktriangle_3(pa1, id2, id3)$	$\boldsymbol{\alpha}(3) = (\boldsymbol{\alpha}_{31}, \boldsymbol{\alpha}_{32}, \boldsymbol{\alpha}_{33})$
$\blacktriangle_4(pa2, mi2, mi3)$	$\boldsymbol{\alpha}(4) = (\boldsymbol{\alpha}_{41}, \boldsymbol{\alpha}_{42}, \boldsymbol{\alpha}_{43})$
$\blacktriangle_5(pa2, ri2, ri3)$	$\boldsymbol{\alpha}(5) = (\boldsymbol{\alpha}_{51}, \boldsymbol{\alpha}_{52}, \boldsymbol{\alpha}_{53})$
$\blacktriangle_6(pa3, pi2, pi3)$	$\boldsymbol{\alpha}(6) = (\boldsymbol{\alpha}_{61}, \boldsymbol{\alpha}_{62}, \boldsymbol{\alpha}_{63})$
$\blacktriangle_7(th1, th2, th3)$	$\boldsymbol{\alpha}(7) = (\boldsymbol{\alpha}_{71}, \boldsymbol{\alpha}_{72}, \boldsymbol{\alpha}_{73})$

The stability grasp index for the spherical, hook and cylindrical grasp can be calculated according to equations:

$$F_S^{Spherical} = \frac{1}{2} \cdot \Big(F_{sgi}\big(\boldsymbol{\alpha}(1)\big) + F_{sgi}\big(\boldsymbol{\alpha}(2)\big) \Big) \tag{5.12}$$

$$F_S^{Hook} = \frac{1}{4} \cdot \sum_{i=3}^{6} F_{sgi}\big(\boldsymbol{\alpha}(i)\big) \tag{5.13}$$

$$F_S^{Cylindrical} = \frac{1}{5} \cdot \sum_{i=3}^{7} F_{sgi}\big(\boldsymbol{\alpha}(i)\big) \tag{5.14}$$

$$\tag{5.15}$$

Note, that the only difference between the hook and the cylindrical grasp is the inclusion of the thumb into the calculation of the stability in the latter case. This choice becomes obvious if we analyze the hook grasp. This grasp is typically used to lift an object with a handle. Gravity plays an important role in these tasks and acts like a virtual finger that generates forces opposing those of the palm. As a result, the object is suffiently stabilized through the interplay of handle, hand and gravity without needing the thumb. In contrast, in the case of the cylindrical grasp, the thumb plays an important role in stabilizing the object in the hand. Without the thumb, we cannot exert sufficient forces on the object in order to lift it.

Fingeropposition Grasps

In this class of grasps, groups of antagonistic fingers apply a force in opposition to each other in order to hold the target object. It includes grasp types such as the tip, lateral and palmar grasp. In case of the lateral grasp, only the thumb is actively exerting force on the object. The other fingers, in particular the index finger, are only passively

fixating the object. An estimation of the stability of such grasps is performed using a combination of the COF and CNO measures introduced earlier.

The palmar and lateral grasp mainly stabilize the object between the thumb and the index finger. During a tip grasp, the object is stabilized through the opposition of the tips of two fingers only. For these grasp types we will compute the stability value based only on two antagonistic sensors. For the palmar and lateral grasp we will use the $th3$ and $id2$ sensors. In contrast to that, we will use the $th3$ and $id3$ sensors for the tip grasp.

Let $\boldsymbol{e}(1)$ and $\boldsymbol{e}(2)$ be the contact points of the two sensors and $\boldsymbol{n}(2)$ and $\boldsymbol{n}(2)$ be the corresponding contact normals. Then, the contact forces exerted by the fingers can be approximated as $\boldsymbol{f}(1) = \boldsymbol{e}(2) - \boldsymbol{e}(1)$ and $\boldsymbol{f}(2) = \boldsymbol{e}(1) - \boldsymbol{e}(2)$. The stability of a fingeropposition grasp can then be estimated by:

$$F_S^{Opposition} = F_{COF}\Big(\boldsymbol{f}(1), \boldsymbol{n}(1)\Big) + F_{COF}\Big(\boldsymbol{f}(2), \boldsymbol{n}(2)\Big) + F_{CNO}\Big(\boldsymbol{n}(1), \boldsymbol{n}(2)\Big) \tag{5.16}$$

The metrics and heuristics discussed so far have low computational demands and allow us to generate estimates of the grasp stability based on a purely geometric analysis. As already described in the beginning of this section, the overall fitness of a given grasp is computed as a sum of the results of the component metrics for stability, distance and anatomical plausibility (see Equation 5.1).

5.4. Grasp Synthesis Algorithm

In this section, we will discuss how the introduced concepts fit together. Specifically, we will disucss how optimization can be used to search for a specific grasp taking into account the learned PLDPM, the fitness function and the target object. We will start with strategies for an intelligent initialization of the optimization parameters. After that, we will discuss the iterative optimization algorithm used for the synthesis process. Finally, we will also discuss computational speedups and efficiency aspects of our grasp synthesis algorithm.

5.4.1. Initialization

Optimization can be a time consuming and computationally demanding process. However, the computation time can often be significantly reduced through the use of simple initialization heuristics. Such heuristics are used to generate an estimation of the starting point of optimization. Ideally, the estimated values correspond to a starting point which is near to a (local) optimum of the fitness function. In this case, the optimizer needs only few iterations to find a good solution.

An estimate for a good wrist position is generated, by combining information about the grasp coordinate system and the size of the palm. More specifically, given the the width w^{Palm} and height h^{Palm} of the palm, the position can be initialzed according to Table 5.2.

Grasp Type	x	y	z
Spherical	$-\frac{1}{2} \cdot h^{Palm}$	0	0
Cylindrical	$-h^{Palm}$	0	0
Hook	$-h^{Palm}$	0	0
Palmar	$-h^{Palm}$	0	$-w^{Palm}$
Lateral	$-h^{Palm}$	$-\frac{1}{2} \cdot h^{Palm}$	0
Tip	$-h^{Palm}$	0	$-h^{Palm}$

Table 5.2: Initialization rules for the wrist position in the grasp coordinate system depending on the generated grasp.

Next, we estimate good initial values for the intrinsic parameters. Empirical evaluations showed that an open hand is often a good starting point. However, in order to make use of this heuristics, we need to determine the initial image coordinates $\widetilde{\boldsymbol{q}}_0$ corresponding to an open hand. To this end, we formulate a simple optimization problem: find a vector $\widetilde{\boldsymbol{q}}_0$, such that $\Phi^{-1}(\widetilde{\boldsymbol{q}}_0)$ corresponds to a maximally opened hand. For cylindrical, spherical and hook grasps, the openening of the hand can easily be determined by the measures found in [Röt07], such as the *'pentagon plane'* approach. For the tip, lateral and palmar grasps we employ a different measure. Precisely, we compute the distance between the tip of the index finger and the tip of the thumb. Using optimization, we try to find a hand shape $\widetilde{\boldsymbol{q}}_0$, such that the distance equals the length of the palm h^{Palm}. Similarly, for the lateral grasp we evaluate the distance of the thumb to the plane spanned by the joints $jid1, jid2$ and $jid4$. Ideally, this distance equals $\frac{1}{2} \cdot h^{Palm}$. Finally, for the palmar grasp, the ideal distance of the thumb to the $jid3$ joint equals h^{Palm}.

5.4.2. Optimization

The goal of optimization is to find a hand shape fulfilling the requirements identified by the grasp quality metric. Different values for the extrinsic and intrinsic parameters need to be tried, until a satisfactory grasp with respect to the metric is found. In turn the metric is dependent on the 3D models used to represent objects in the virtual environment.

Such 3D models are specified in terms of polygon-based approximations of physical objects. Hence, they generally do not have a smooth, continuous surface. We also cannot assume that the fitness function defined by the metric is continuously differentiable. Optimization techniques based on the computation of gradients are therefore not applicable

to this problem. On the other hand, because an object can be grasped in many ways, we can expect the fitness function to have many local optima, i.e. it is *multimodal*. We therefore need an optimization algorithm that can deal with non-differentiable, multimodal fitness functions.

These properties are met by a special type of evolutionary algorithms called evolution strategies (ES). ES represents each solution candidate (or chromosome) as a floating point vector. Selection in ES can be done in two ways, the so-called *comma*-selection and the *plus*-selection. According to the notation used in EA literature, the two selection operators can be written as (μ, λ) and $(\mu + \lambda)$. The variable μ denotes the number of parents, whereas λ denotes the number of offspring to produce. In plus-selection, the parents compete with their children for a place in the next population. In comma-selection, only the children compete among each other. The competition is won by the μ chromosomes which have the highest fitness. Mutation is considered to be the main search operator in ES, whereas reproduction is typically neglected and will not be used in the following. The mutation step adds random noise sampled from a normal distribution to the variables of a chromosome. A distinguishing feature of ES is that chromosomes do not only carry parameters specific to the target problem, but also strategy parameters of the ES itself. In our case, we will encode the step size σ of the mutation operator into the chromosome. Hence, the mutation operation can be written as:

$$\sigma \leftarrow \sigma \cdot \ln\left[\mathcal{N}(0,1)\right] \tag{5.17}$$

$$x \leftarrow x + \sigma \cdot \mathcal{N}(0,1) \tag{5.18}$$

The control parameters like σ are evolved according to the rules which govern the ES. This feature of ES is often termed *self-adaptation*. It removes the necessity to exogenously define the control parameters, e.g. by a user, or a fixed schedule.

In the following we will use a particular type of ES called encapsulated evolution strategies (EES) [GUS99]. EES is a nested variant of evolution strategies, in which an outer (high-level) evolution process repeatedly triggers a set of inner (low-level) processes. Thus, the method performs a combination of local and global search. Each of the λ^{global} offsprings generated by the high-level evolutionary process is used as a parent chromosome for an isolated population of μ^{local} individuals. Each of the generated populations is then evolved through a separate low-level ES for γ^{local} generations. Once the low-level processes are finished, the fittest individuals are propagated back to the high-level process. The best individual among these solutions is then selected and used to spawn a new set of low-level populations. This is repeated for γ^{global} generations.

An algorithmic description of the grasp optimization procedure is given by Algorithm 4. First, both extrinsic and intrinsic parameters of the grasp are initialized according to the scheme explained in Section 5.4.1. The initialization process uses simple heuristics such as to avoid starting from a bad area of the search space. Then a first chromosome $\boldsymbol{x}$ is created. The chromosome includes the extrinsic parameters $\boldsymbol{p} \in \mathbb{R}^3$ (position of

Algorithm 4 An algorithm for grasp optimization based on encapsulated evolution strategies. It outputs the image coordinates of the grasp $\widetilde{\boldsymbol{q}}$, the wrist orientation $\boldsymbol{\alpha}$, and the position of the hand $\boldsymbol{p}$.

Require: $\gamma^{local}, \gamma^{global}, \lambda^{local}, \lambda^{global}$
1: {*Part 1: Parameter Initialization*}
2: $t^{global} \leftarrow 0$
3: $(\boldsymbol{p}, \boldsymbol{\alpha}, \widetilde{\boldsymbol{q}}) \leftarrow$`initialize`$(\boldsymbol{p}, \boldsymbol{\alpha}, \widetilde{\boldsymbol{q}})$
4: $\boldsymbol{x} \leftarrow (\boldsymbol{p}, \boldsymbol{\alpha}, \widetilde{\boldsymbol{q}}, \sigma^{global}, \sigma^{local})$
5: {*Part 2: Global optimization loop*}
6: **while** $t^{global} < \gamma^{global}$ **do**
7: $\quad\mathcal{E} \leftarrow \emptyset$
8: $\quad\mathcal{P}^{global} \leftarrow$ create λ^{global} copies of $\boldsymbol{x}$
9: $\quad\mathcal{P}^{global} \leftarrow$ `mutate`$(\mathcal{P}^{global})$ {*mutate chromosomes using* σ^{global} *of each individual*}
10: $\quad$**for all** $\boldsymbol{x'} \in \mathcal{P}^{global}$ **do**
11: $\quad\quad t^{local} \leftarrow 0$
12: $\quad\quad$ {*Part 3: Local optimization loop*}
13: $\quad\quad$**while** $t^{local} < \gamma^{local}$ **do**
14: $\quad\quad\quad\mathcal{P}^{local} \leftarrow$ create λ^{local} copies of $\boldsymbol{x'}$
15: $\quad\quad\quad\mathcal{P}^{local} \leftarrow$ `mutate`$(\mathcal{P}^{local})$ {*mutate chromosomes using* σ^{local} *of each individual*}
16: $\quad\quad\quad$`evaluate`$(\mathcal{P}^{local}, F, <)$ {*calculate the fitness value for each individual*}
17: $\quad\quad\quad x' \leftarrow$ `select`$(\mathcal{P}^{local}, <)$ {*select individual with lowest fitness*}
18: $\quad\quad\quad t^{local} \leftarrow t^{local} + 1$
19: $\quad\quad$**end while**
20: $\quad\quad\mathcal{E} \leftarrow \mathcal{E} \cup \{x'\}$ {*add best individual from current run to set of elites*}
21: $\quad$**end for**
22: $\quad x \leftarrow$ `select`$(\mathcal{E}, <)$ {*select individual with lowest fitness*}
23: $\quad t^{global} \leftarrow t^{global} + 1$
24: **end while**
25: **return** $(\boldsymbol{p}, \boldsymbol{\alpha}, \widetilde{\boldsymbol{q}})$ {*intrinsic and extrinsic parameters of the synthesized grasp*}

the hand) and $\boldsymbol{\alpha} \in \mathbb{R}^3$ (orientation of the hand), as well as the intrinsic parameter $\widetilde{\boldsymbol{q}} \in \mathbb{P}$ (shape of the hand). The chromosome also includes the parameters σ^{global} (global mutation step size) and σ^{local} (local mutation step size).

The *outer* optimization loop generates a set of *inner* evolutionary processes. At the end of each of these processes, the individual with the best fitness is added to the *elite*-list $\mathcal{E}$. This list contains all grasps that have been generated in the local evolutionary processes. After λ^{global} local ES have been run, the best grasp in the elite-set is determined. This grasp is then used as a starting point for the next set of local ES. Note, that the computation of the fitness values for each grasp requires a learned PLDPM, as well as the geometry of the object to be grasped. After termination, the algorithm returns the best individual recorded in the elite-list. The combination of global

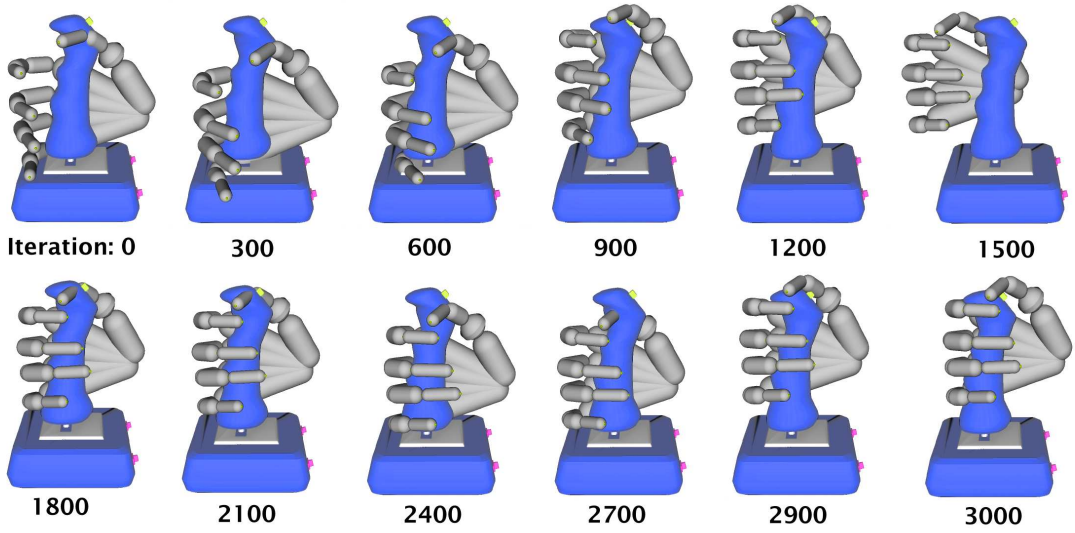


Figure 5.8: Visualization of the optimization process. The hand position and posture is varied until a satisfactory grasp with respect to the fitness function is found.

optimization with local fine-tuning allows for a good tradeoff between exploration and exploitation. We can explore the search space for different types of solution candidates, while we are at the same time locally optimizing the parameters of each of these potential solutions. This can easily be understood, by taking a look at Figure 5.1. Each of the grasps shown in the gray box is a solution from a local ES. On a local scale, the parameters of each of these grasps is optimized, in order to avoid penetrations or other unfavorable characteristics. Then, on a global scale, the results of the local ES are collected and a single overall best grasp is determined.

Figure 5.8 shows an example of the evolutionary optimization process. As we can see, the initialization heurisitics ensure that even at iteration zero the hand is close to the object to be grasped. Throughout the course of optimization, the position, orientation and shape of the hand are adapted until they slowly converge to the ideal grasp.

5.4.3. Computational Speedups

A simple and efficient way to speed up the grasp synthesis algorithm is to use a low-polygon version of the object model for the optimization process. For graphical display, however, the original model can still be used. For example, when optimizing a grasp for the Stanford bunny, we use the high resolution version with 26332 polygons for display and a low-polygon version with 453 polygons for optimization. In general, however, it is also important that the shapes of the original model and the reduced model do not deviate too much. Another way to speedup computation while at the same time increasing the control over the optimization process, is the specification of an *area of*

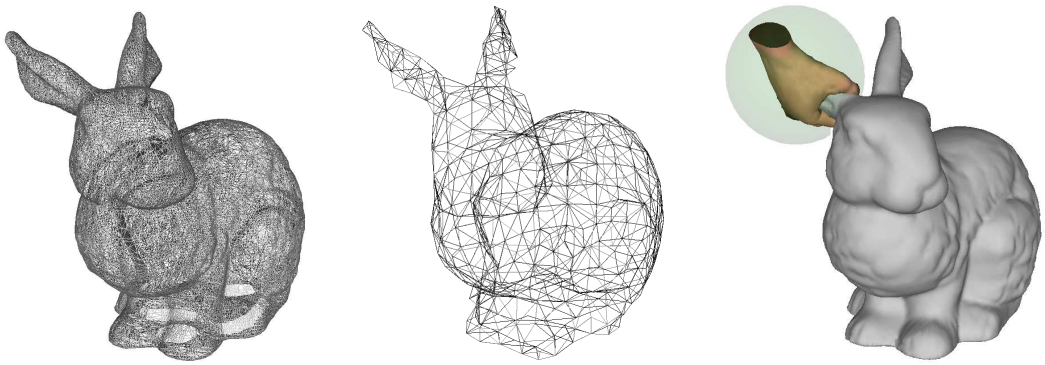


Figure 5.9: Left: Visualization of the polygons used for modeling the Stanford bunny. Middle: The low-polygon version is used for fast optimization. Right: The definition of an *area of interest* (green circle) specifies which part of the object should be grasped.

interest. This allows the user to define a specific part of the object within which a grasp should be generated. Figure 5.9 (right) shows an example for a spherical area of interest. In this example, we want the optimizer to find a grasp which grasps the left ear of the bunny model. Therefore, we placed the area of interest in a way that it envelopes all triangles of the left bunny ear. All polygons outside of the area of interest are discarded from the optimization process. In our specific case this reduces the number of polygons to evaluate from 453 to 120 polygons. Generally, using an area of interest leads to noticeably lower computational demands.

5.5. Evaluation and Results

The quality of the grasp synthesis algorithm proposed is affected by the correct choice of its parameters, such as the recorded grasps, the employed DR method, and the rotation representation. An important goal of the conducted evaluation, is to find *'ideal'* values for these parameters. Another goal of the evaluation is a better understanding of the dependency between the quality of generated results and particular parameter settings. Therefore, a set of incremental experiments was performed. The word *incremental* refers to the fact that the results of anterior experiments are incorporated into consecutive experiments. For example, if a particular DR technique is found to yield the best results in experiment A, it is used as the DR method of choice in the subsequent experiment B. Thus, with progressing experiments, more and more parameters are set to fixed values. Overall, four experiments $E1 - E4$ have been conducted, where each experiment focused on a particular parameter or variable of the grasp synthesis algorithm. The following sections present each of these experiments as well as the corresponding parameter settings.

5.5.1. E1: Data Acquisition and Analysis

During data acquisition, ten subjects were asked to provide demonstrations of natural grasping hand shapes. An optical *fingertracking* system by A.R.T. [ARTG10] was used as capturing device for hand shape data. The system tracks movements of all 5 fingers of the hand and comes with software that computes, among other data, the position and orientation of the hand, the fingertip positions, and the rotations of finger joints. Before application, the fingertracking system was calibrated in order to estimate the size and parameters of the user's hand. Calibration was performed using the vendor provided software tool and typically takes about two minutes per user.

The ten subjects were asked to perform grasps of the Schlesinger taxonomy using the fingertracker. For each grasp type the data was recorded seperately, yielding six different sets of observation data per user. During a recording session, each user was asked to grasp various physical objects. Hand shape data was collected continuously, both during the closing and the opening phase of the hand. This means that the hand shape database is not restricted to *'peak'* hand shapes where an object is fully grasped. The typical duration of the data acquisition phase was about 15 minutes. Hand poses in the database are stored as rotations of finger joints (3 ball joints per finger, i.e. 45 degrees of freedom in total). The first post-processing step is to transform the joint rotation data into an *exponential map* representation [Gra98]. The result is a set of 45-dimensional vectors, each of which represents a single hand posture. The reason for using the exponential map representation is that it transforms the rotation into a linear space. It was argued in [EMMT04] that the application of DR techniques benefits from such a representation. However, this hypothesis will be investigated in more detail in experiment $E3$.

Next, the recorded data sets were processed in order to construct corresponding PLDPMs. Different types of DR techniques were applied on the data sets. For each user and each grasp type, a different PLDPM was learned. The goal of this experiment was to find answers to the following questions:

- How well do the different DR techniques compress the experimental data?
- How does the recorded data vary among the different grasp types?
- How does the recorded data vary among the different test subjects?

Figure 5.10 shows the result of applying different DR techniques on recorded grasps. The figure depicts the angular difference between the original hand shapes and their reconstructed counterparts after applying a particular DR technique. Along the x-axis are the different values of dimension L of the low-dimensional space. We observe for PCA, MDS, and NLM that the projection error becomes smaller with increasing value of L. This is not the case for the other DR techniques. For example, the LLE algorithm shows a peak error when projecting into a space with 12 dimensions. This can be

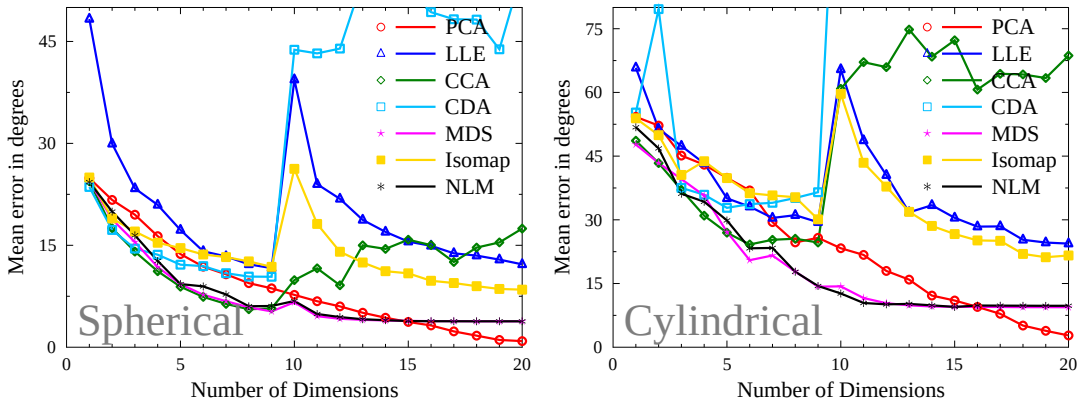


Figure 5.10: Result of applying different dimensionality reduction techniques on recorded grasps. The number of dimensions L on which to project was varied and the reprojection error was measured. The y-axis shows the error (in degree) for a full pose.

explained by numerical problems occuring when the number of dimensions equals the number of neighbors used for the local embedding. Still, we can see in the graphs, that for all grasp types the recorded data can be projected onto a few dimensions without loosing much of the information contained. For example, in the case of the cylindrical grasp, an error of about $35° \sim 50°$ is introduced by projecting onto a low-dimensional space with $L = 3$. Divided by all joints of the hand, this translates to an error of about 2°to 3° per joint.

To better understand this result, Figure 5.11 (left) summarizes the projection error (computed over all test subjects) when projecting onto a three-dimensional space. Obviously, the best compression of the data was achieved using CCA. MDS, Isomap and NLM stay significantly below the barrier of 30°. Using these techniques we can keep the per-joint reconstruction error below 2°. It is noteworthy though, that MDS, a linear DR technique, achieves competitive results compared to the newer and more sophisticated NLDR techniques.

Figure 5.11 (right) shows the projection error for different grasp types when performing a PCA with $L = 3$. We see that the projection error varies significantly among the investigated grasp types. The smallest error ($\approx 19°$) was achieved for the tip grasp. This can be explained by the fact, that the tip grasp only involves two fingers and does not leave much room for variation. The highest error ($\approx 75°$) can be observed for the hook grasp. This might be explained by the fact that during a hook grasp several markers on the fingertracking device can not be monitored by the tracking cameras anymore. As a result, the recorded hand shape information gets corrupted. Another grasp with a high projection error is the lateral grasp. This is surprizing because the lateral grasp, similar to the tip grasp, does not involve many active fingers.

In summary, hand postures can be efficiently compressed using DR techniques. In this regard, our observations are consistent with the results found in [SFS98]. Also, we

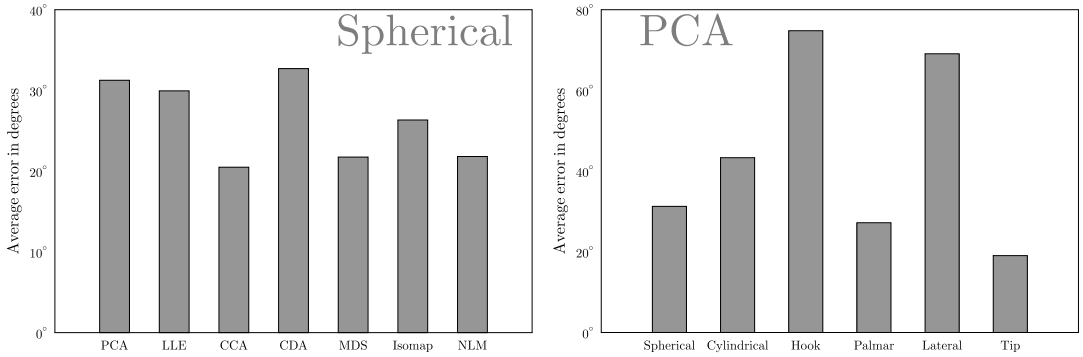


Figure 5.11: Reprojection error of poses (in degree) introduced through the application of dimensionality reduction and projection on three dimensions. Left: Results of different dimensionality reduction techniques applied on the same data set. Right: Results of PCA on data sets corresponding to different grasp types.

observed that CCA, MDS, Isomap and NLM seem to be slightly better suited for this task than the other DR techniques. Further, the performance of the DR techniques is dependent on the variability inherent to a particular grasp type. Grasp types with low variability, such as the tip grasp, can be represented using a smaller number of principal components.

5.5.2. E2: Simple optimization setting

The PLDPMs learned in experiment $E1$ are used to synthesize grasps on virtual objects. For a first evaluation of the results, only PLDPMs learned with PCA were used. This allows for a first, thorough analysis of the synthesis algorithm in a restricted setting.

The synthesis algorithm was run for PCA-based PLDPMs on the set of twenty-one virtual objects, i.e. a bottle, a coffe jug, a table, a screwdriver, a hammer, a joystick, a case, a computer mouse, a CD, a matchbox, a key, a suitcase, a card, a pencil, a taperoller, a bag, a rubber, a Tennis ball, the Stanford bunny, a toy, and a nail (see Appendix A). For each grasp type, six grasps with different objects were generated, resulting in 36 grasps per test subject. The resulting grasps were then automatically classified into *good*, *acceptable*, and *bad* according to their fitness. A threshold κ was introduced for this purpose and all grasps that yielded a fitness smaller than $\frac{1}{2} \cdot \kappa$ were regarded *good* grasps. Grasps in the interval $[\frac{1}{2} \cdot \kappa, \kappa]$ were labeled *acceptable*. Finally, all grasps with a fitness higher than κ, were deemed *bad*. Appropriate values for κ can be derived by computing the sum of the weights of the sensors involved, the stability term, as well the anatomical plausibility term. The reason for this, is that each term of the fitness function is configured to generate a value smaller than or equal to one for grasps that are acceptable or good. This is a direct consequence of the violation function V used here. Table 5.3 shows the grasp dependent values of κ and how they are computed as a sum of the different weights.

	Weighting of fitness components			
Grasp Type	F_D	F_S	F_A	κ
Spherical	6	1	1	8
Cylindrical	9	1	1	11
Hook	11	1	1	13
Palmar	6	3	1	10
Lateral	4	3	1	8
Tip	2	3	1	6

Table 5.3: Calculation of the value κ as a sum of weights for the individual components. Because the number of sensors varies depending on the grasp type, the resulting fitness values can have different value ranges. κ scales the results to the same range and is calculated based on the number of sensors and components involved in the fitness computation.

In our experiment, we employ the automatic classification in order to address the following imporant questions:

- What is the percentage of good grasps among all grasps generated?
- Does the quality of the synthesized grasps vary among different grasp types?

In order to ensure a fair comparison among the grasp types, we classified the results of each type according to table 5.3. For each grasp type, each of the 60 synthesized grasps (6 objects $\times$ 10 subjects = 60 grasps) is classified as good, acceptable or bad. The results are presented in Figure 5.12. Although we incorporated the intrinsic difficulty of each grasp type into the analysis, the hook grasp is still among the most difficult to synthesize. In the case of this grasp, about a third of the results are classified as bad. In comparison to that, in the case of the lateral grasp, only good or at least acceptable grasps were synthesized. Similarly, in the case of the cylindrical and spherical grasps only a small number (< 5) of grasps were found unusable. For cylindrical, spherical, lateral and tip we observe that at least 50% of all synthesized grasps are good. As expected, the lowest number of good grasps (14) was achieved for the hook grasp. Still, if we consider all grasp types, good and acceptable grasps represent more than 85% of the results.

In summary, in this analysis we found that the complexity of synthesizing new grasps varies among the grasp types. Among the most complex grasps to synthesize are hook grasps. At the same time, we find that about 85% of the grasps generated in this setting were acceptable or better.

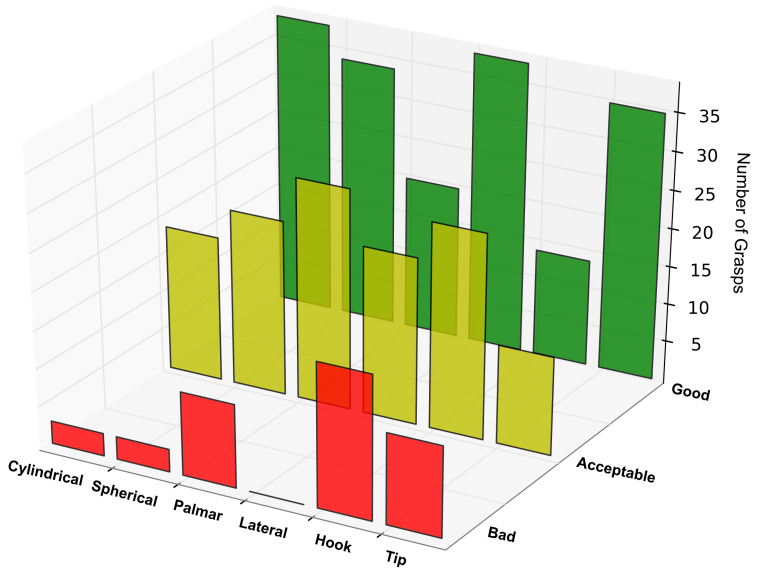


Figure 5.12: Classification of synthesis results into good, acceptable and bad grasps.

5.5.3. E3: Optimization with different Rotation Representations

In the next experiment we investigated the choice of rotation representation. As already explained, rotations can be specified using different representations. It was argued in [EMMT04] and [SCF06] that an exponential map representation is best suited when using DR techniques. However, to our knowledge, there is no empirical evidence backing up this hypothesis. Therefore, in this experiment, we repeated experiments $E1$ and $E2$ for different rotation representations. For this, we transformed each recorded data set into a different rotation representation and learned corresponding PLDPMs. Then, the learned PLDPMs were used to synthesize grasps as explained in experiment $E2$. Most importantly, we tried to understand whether the choice of rotation representation affects the optimization results.

Using the automatic classification introduced in experiment $E2$, five rotation representations, axis-angle, Euler, exponential map, matrix, and quaternion were compared. Figure 5.13 shows the results of these experiments. The green, yellow and red colored segments depict the ratio of good, acceptable and bad grasps, respectively. We can see that there is a substantial difference in the results. The axis-angle representation, as well as the Euler representation produce about 40% good grasps and 40% acceptable grasps. Hence, with these representations an overall success rate of about 80% is achieved, where good and acceptable grasps are equally probable. In contrast to that, the exponential map, matrix or quaternion representation all achieve an overall sucess rate of about 85%. A more detailed analysis of these results shows that the quaternion representation out-

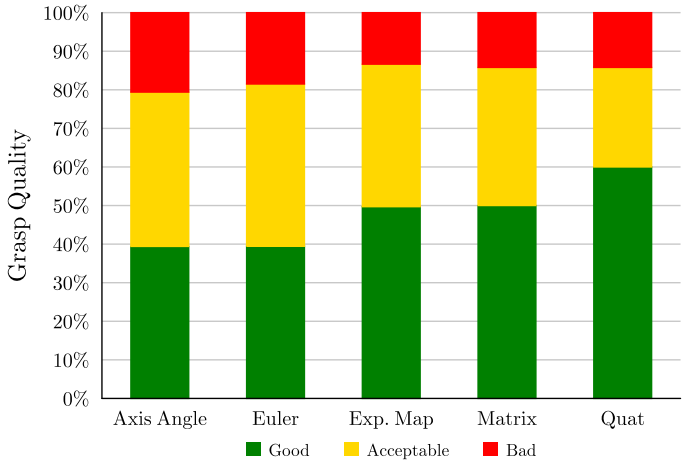


Figure 5.13: Percentage of good, acceptable and bad synthesized grasps depending on the used rotation representation.

performs all other representations used. In particular, quaternions result in 60% good grasps compared to 50% or 40% as achieved by the other representations.

Returning to our original question, the experiment reveals that the choice of rotation representation substantially influences the results of the optimization algorithm. Hence, it should be chosen with care. The exponential map representation, together with matrix representation and quaternions is well suited for our application domain. At the same time, the results indicate that using quaternions improves the quality of the results as compared to all other rotation representations. While the overall success rate of 85% from experiment $E2$ could not be improved, there is still a significant difference in the percentage of good grasps generated. When quaternions were used, about 60% of the synthesized grasps were high-quality grasps.

5.5.4. E4: Optimization using different DR techniques

In this experiment the effect of different DR techniques on grasp synthesis was evaluated. A similar experiment as in $E2$ was performed on the data set of one of the test subjects. In contrast to $E2$, however, the experiment was repeated with different DR techniques. Additionally, the rotation representation was chosen based on the results of experiment $E3$. The aim of the experiment was to answer the following questions:

- Does the choice of the DR technique affect the quality of the synthesized grasps?
- Does a good fitness value also lead to a visually appealing grasp?
- How much time is needed to successfully synthesize a grasp?

The experiment is performed with PLDPMs learned using different DR techniques. Following the insights gained by experiment $E3$, quaternions were adopted for specifying the rotations. Running the optimization algorithm on each of these settings resulted in 36 grasps per DR technique. Each of the grasps was then automatically labelled according to the scheme introduced earlier in Section 5.5.2. Table 5.4 summarizes these results.

	Grasp Synthesis Results			
	Good	Acceptable	Bad	Av. time/s
CCA	26	6	4	17.99
MDS	24	7	5	18.01
NLM	23	9	3	17.75
PCA	23	8	4	2.62
Isomap	19	9	8	17.75
LLE	18	14	4	17.80
CDA	18	13	5	18.00

Table 5.4: Classification of the synthesized grasps into good, acceptable and bad results.

The results suggest that the choice of the DR method also has an important influence on the optimization process. More specifically, CCA, MDS, NLM and PCA seem to be well suited for the task, as they generate 23 to 26 good grasps for this specific user. In contrast, the Isomap, LLE and CDA do not perform well for the current task. The best result, 26 good grasps, is achieved by the CCA algorithm. The latter results are in close accordance with the results of experiment $E1$, where CCA, MDS and NLM already showed good performance in compressing the data.

Now, if we turn to the runtime of each of the settings, we find that except PCA all DR methods have an average runtime of about 18 seconds per grasp. The bottleneck is Algorithm 1, which is used to reconstruct a posture from its low-dimensional image point. It requires searching for K nearest neighbors of the image point, a computationally demanding step. In contrast to that, by using PCA a grasp can be synthesized in about 3 seconds. This can be explained by the fact, that reconstruction of postures with PCA is a simple matrix multiplication. Figure 5.14 shows several examples of the synthesized grasps and the corresponding classification.

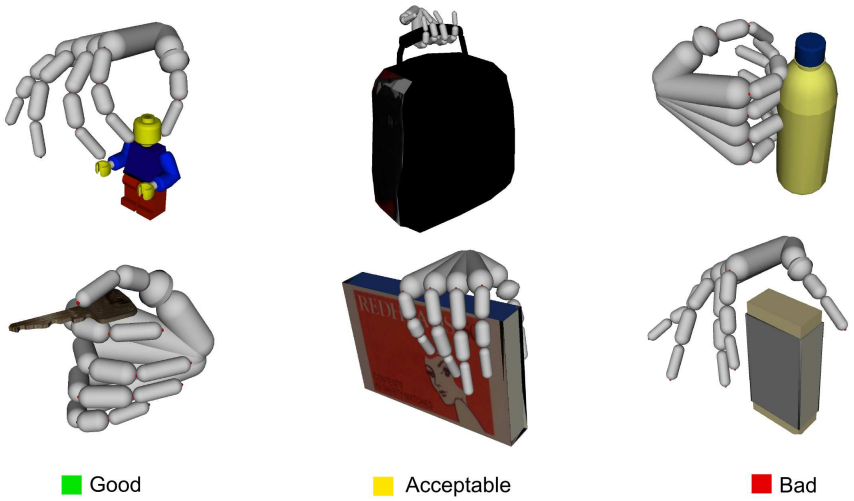


Figure 5.14: Results of automatic classification of synthesized grasps into *good*, *acceptable* and *bad* grasps, based on the introduced classification scheme.

5.6. Other Approaches

By now, there is a considerably body of research on realistic virtual grasping. Generally, the explored approaches can be divided along several criteria axes. Some approaches use an underlying physical simulation of the human hand, whereas others solely rely on graphical geometry. We will refer to the former as physics-based or simulation-based, e.g. [BI05; PZ05; KP06], whereas the latter will be referred to as geometry-based [ST94; WHAJ06; HUWK07]. Another criterion is whether data of grasp examples (generally, motion capture data) is used for the grasp synthesis process or whether the grasps are the product of a set of controller functions or algorithms (model). The former type of approach is often called data-driven or example-based, e.g. [LFP07] whereas the latter one is called model-driven or behavioral, e.g. [ST94; KT99; KL00; DLB96; RG91]. Recently, several hybrid approaches have emerged which strive to combine the strenghts of the various types while compensating the weaknesses.

A classical geometry-based and model-driven approach is described in [ST94]. Objects are represented as one of three primitive shapes which are then mapped to an appropriate grasp in a corresponding taxonomy. Fingers are closed until contact between the geometric hand representation and the object occurs. Limitations of this approach are the not very lifelike movements, the absence of real interaction with virtual objects and the restriction to primitive shapes.

Simplifications based on primitives are also employed in modern grasp synthesis algorithms. In [GHBK09], the object is first decomposed into a set of bounding volumes. Then, edge-detection methods are used to find good grasp regions on the surface of the object. A similar approach is used in [GALP07] and [XKZD09]. Given a 3D object,

first, a so-called *'decomposition tree'* is generated which is a hierarchical approximation of the object's shape using simple primitives. This tree is then used to prune the space of possible grasps and, hence, render the synthesis process more tractable. Approaches that are based on this kind of decomposition bear various drawbacks, as each new object has to go through a pre-processing phase before it can be used. Additionally, because of the approximation involved, the grasps cannot be adapted to fine details of the object's surface. Hence, these approaches are mostly restricted to coarse types of grasps.

A simple and flexible geometry-based approach for hand-object interactions in virtual environments is presented in [HUWK07]. It aims at a visually feasible output and not a realistic simulation. A simple grasp condition based on an arbitrary number of sensors yields an object transformation based on contacts with the spherical sensors of the virtual hand. Multiple objects can be manipulated by multiple fingers at the same time. However, since there is no real dynamics simulation, no finger manipulation beyond grasping is possible, e.g. pushing with a single finger.

Recently, the integration of physics-based approaches has become more widespread, since they overcome some of the limitations mentioned above and produce more realistic looking interactions. In [PZ05] a physics-based approach is outlined, where controller parameters are derived from a database of example grasps. The goal was to create hand motions similar in quality to motion capture data, but usable for varying object sizes and shapes as well as different hand sizes. The proposed controller combines passive and active control which enables the hand to be influenced by moving objects in addition to the hand manipulating scene objects (rigid bodies) in a physically plausible way. The hand is controlled by calculating torques for each of the finger joints. Although general, relatively simple and consistent for movements with or without contact, the approach still has some limitations. No automatic hand orientation is done and the type of grasps supported is limited to cylindrical enveloping hand shapes.

Some of these limitations are addressed in [LFP07]. Again, backed by a grasp database, several possible ways to grasp an object are identified and evaluated with respect to a given grasp quality metric. A shape matching algorithm is introduced to identify the hand orientation and finger pose towards the object, so that the grasp fulfills requirements for a given user-specified task. These requirements refer to forces applied to the object and are evaluated on the basis of an anatomical hand model involving tendons and muscle force limits. While providing a flexible set of plausible grasps, still some limitations remain. The approach is not realtime capable (runtimes of several minutes to identify candidate graps), only semi-automatic (the user has to pick from the presented set of candidates) and limited to hand models which are very similar in size to the captured hand in the database. Again, the type of grasps is limited to enveloping power grasps.

Instead of guessing or specifying hand compliance parameters, in [KP06] contact force parameters are measured during user interactions with (real) objects via specialized hardware. This enhanced version of motion capture is referred to as interaction capture. An intermediary representation in form of an interaction trajectory containing motion as

well as contact forces is generated. On this basis grasping animations can be synthesized involving virtual objects with variations in shape size and physical parameters. The employed hand model is a regular kinematics chain with added compliance values at the joints while the movement of the virtual objects is simulated with a rigid body dynamics simulation.

In terms of the categorization above, our approach falls in the category of geometry-based, data-driven approaches. The aim is to find a good (w.r.t. a given grasp quality metrics) positioning of the hand and the fingers on an arbitrary virtual object while maintaining the style and naturalness of the human-demonstrated examples. As such our goal is related to that adressed in [LFP07]. However, our approach is automatic, fast enough to be used in interactive scenarios and can be used for arbitrary types of grasps (w.r.t. a given grasp taxonomy). Hence, it fulfills the demands of virtual reality applications, especially *action capture* as described in [JAHW06].

5.7. Conclusion

In this chapter, we presented a new method for the imitation of human grasping abilities. The proposed approach is based on motion capture data recorded from human subjects, which is used to train a PLDPM. Possible hand postures assumed during grasping actions are encoded this way. Grasp synthesis is then realized by searching the grasp space for a hand shape that optimizes the given grasp quality metrics. The grasp spaces encoded by the PLDPM can be searched through efficiently, while being large enough to contain a variety of plausible grasps.

The experiments revealed that CCA, MDS, NLM and PCA are well suited for the low-dimensional compression of grasps as well as the later synthesis thereof. The best performing dimensionality reduction technique with respect to the quality of synthesized grasps is CCA. However, PCA achieves a better tradeoff between computational demands and quality of results. With PCA most grasps were synthesized in about three seconds, which amounts to $\frac{1}{6}$ of the time needed by the other methods. The experiments also show that the type of rotation representation used can have a strong influence on the result. The best results are achieved using a quaternion representation. Finally, the experiments also show that about 85% of the grasps generated by the grasp synthesis algorithm are acceptable or better.

A strong point of the algorithm is the fact that it does not require any preprocessing of the object to be grasped, i.e. any arbitrary 3D model can be used. Other approaches to automatic grasping reported in the literature (e.g. [XKZD09]) often require a preprocessing step in which the user or an algorithm decomposes the object into basic shapes. This is labour intensive and typically results in a crude approximation of the original shape. Such a decomposition becomes particularly difficult, when dealing with curved, smooth objects. Similarly, other approaches use preprocessing steps, such as clustering, shape-matching or pruning in order to generate a database of viable grasps from which to

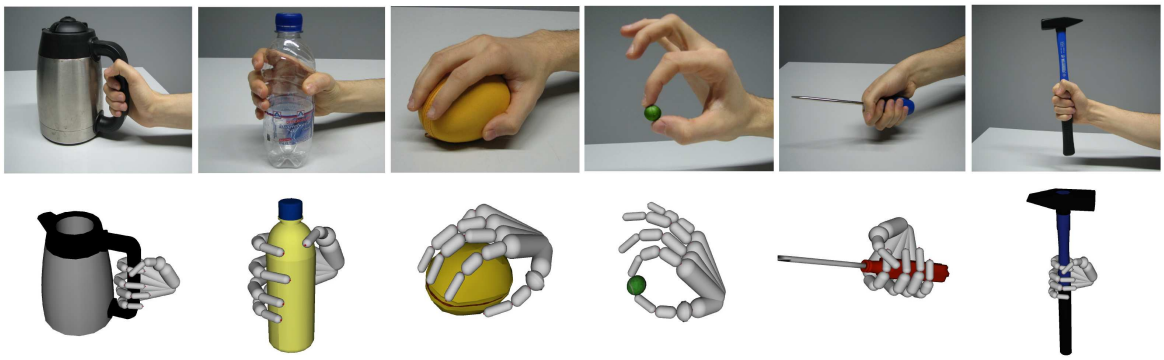


Figure 5.15: A comparison between human hand shapes and synthesized grasps.

choose later. All these steps are not necessary for the algorithm presented here. Another strength of our algorithm is the underlying data-driven approach. A new PLDPM can be trained within minutes, by recording demonstrations via motion capture. This way, the appearance of the generated grasps can easily be adapted to the anatomical features of a particular subject without having to change the synthesis algorithm. Hence, shape and style of the synthesized grasps imitate those of the human teacher. In Figure 5.15 we see comparisons between recorded and synthesized grasps. The results produced by the algorithm closely resemble the human demonstrations.

One aspect that needs further improvement is the computation time. Using PCA as DR technique, a grasp can be generated within a few seconds. While this is orders of magnitude faster than results reported in other publications, it is still not sufficient for real-time environments such as video games. The temporal bottleneck here is the computation of distances between the sensors and the object. One idea for speeding up this process, is to perform these computations on the graphics card, i.e. to develop dedicated vertex shaders. The calculations could then be performed in parallel, resulting in a substantionally lower computation time. Yet, even in its current form, the grasping algorithm is well suited for a wide range of application domains. For instance, it can be used for animation of grasping interactions in motion picture productions. Here no strong real-time requirements need to be met.

For a fine control over the type of the grasp generated, the animator is provided with four principal means of parameter adjustment:

1. an approach direction and orientation of the hand can be specified
2. a region of interest can be defined, i.e. the part of the object to grasp
3. a specific PLDPM can be trained and applied to control the grasp type
4. different grasp stability criteria can be employed in the optimization method to fine-tune hand–object contacts

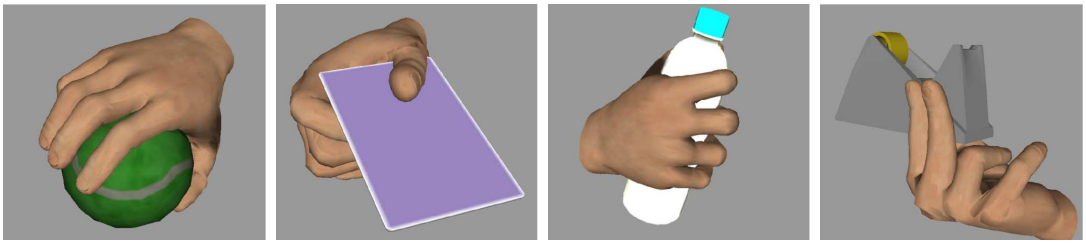


Figure 5.16: Grasps from the Schlesinger grasp taxonomy synthesized using the introduced optimization approach. From left to right: spherical, lateral, cylindrical and palmar grasp.

The training of specific grasp models was demonstrated using the Schlesinger taxonomy (see Figure 5.16). Clearly, the method can also be applied to other grasp taxonomies. The animator might even choose to train highly specific grasp models for individual objects. When physical objects are grasped, further degrees of freedom of the hand shape may be created by the contact between the hand and the object. If a high degree of realism is required for the animation, such effects could be reproduced by training models from grasp demonstrations on physical objects only.

As grasp stability measurement, the method has been tested with purely geomteric criteria for estimating the grasp quality. This decision was intentionally made so as to free the animator from the necessity of modeling physics-related object properties. The presented method itself does, however, not depend on a particular optimization criterion and is compatible with more complex grasp quality measures as researched, e.g. in the GraspIt project [MC03].

The presented grasp synthesis algorithm is also an important component of the action capture method [JAHW06] which will be presented in Chapter 7. Based on the algorithm presented here, action capture allows virtual humans to imitate object manipulations in changing environments. As we will see in Chapter 7, this approach is particularly attractive for virtual prototyping applications.

6. Learning to Imitate and Adapt Full-Body Motions

6.1. Introduction

First, we have applied our approach to single finger motions, and then moved on to focus on the generation of grasping behavior for an entire hand. In this chapter, we will investigate whether the proposed approach scales up to the imitation of motions involving the whole body. The synthesis of full-body motions, like walking or other locomotion tasks, is a key requirement for our target application domains.

Simultaneously, we try to explore different variations of the imitation learning setting used in this thesis. More precisely, we will progressively perform the following modifications to our standard setting:

1. employ different synthetic humanoids
2. synthesize behaviors in dynamic environments
3. explore different means of conveying demonstrations

Especially for applications in the domain of robotics it is important to show that the proposed approach can synthesize motions for different implementations of synthetic humanoids. We must be able to generate motions for virtual humans and robots of different sizes and with different physical properties. Therefore, in the following sections we will provide examples of behaviors synthesized for a set of different virtual humans, a simulated humanoid robot, a real humanoid robot, as well as a sophisticated android robot.

The second point of the list above refers to the environment in which the behaviors are synthesized. So far, we only considered virtual worlds that were based on geometric modeling; the aspect of *dynamics* was left out. However, our real world is governed by the laws of physics. As a result, physical phenomena such as gravity or friction have a strong impact on human motion and behavior. Standing up, keeping balance, and walking are examples of behaviors that result from the interplay of internal forces exerted by our muscles and external forces originating from the environment. In Section 6.3 we will discuss an imitation learning setting in which physics is modeled.

Finally, the last point of the list refers to the way demonstrations and example motions are recorded. To this day, motion capture has been the predominant approach

for recording motions in imitation learning scenarios. Motion capture has several features that make it very attractive for this purpose. Most importantly, it enables us to record demonstrations with a high degree of precision and without the need for complex postprocessing. At the same time, motion capture bears many disadvantages, such as high costs for tracking devices and spatial requirements. One of the most important limitations of motion capture, however, is the limited mobility; motions can only be recorded at dedicated special-purpose facilities. To overcome these problems, other, less demanding forms of motion recording can be used. In this chapter we will explore two slightly different approaches to this end. In Section 6.3 we will use a technique that is reminiscent of puppeteering. Herein, a robot is kinesthetically manipulated by a human user. The robot is only passively involved in the recording process. In contrast to that, in Section 6.4 we will present a method that is based on the physical interaction between a human and an android robot.

6.2. Imitation from Motion Capture Data

In this section, we will investigate the use of motion capture data for learning locomotion skills. Locomotion is, like grasping, an essential skill for virtual humans. In order to inhabit a virtual environment in a realistic and meaningful way, virtual humans need to go from one place to another. For this, they might need to climb stairs, walk, run or perform evasive side steps. During the creation of a virtual environment, e.g. for a video game, the programmer must anticipate these situations and provide the game character with routines that generate the corresponding skill.

However, producing realistic simulated walking and locomotion skills is an ongoing topic of research. Currently, popular approaches to the synthesis of walking skills use prerecorded motions which are rearranged in longer animation sequences. Such approaches do not adequately reflect the dynamic nature of locomotion. In order to produce convincing results, a simulated locomotion skill must respond to properties of the current terrain, such as gaps, obstacles or height differences. Optimization-based approaches to simulated walking can dynamically adapt the generated motion to the current context. This flexibility often comes at price: the gaits produced appear unnatural. The difficulty here lies in the fact that the human visual system is highly sensitive to biological motion. This sensitivity allows us to extract information about gender, intentions and emotions of an observed person. Even small imperfections in the generated animation can have an uncanny effect on the viewer. Hence, the generation of simulated locomotion skills must entail both the adaptation to a new context as well as the production of convincing, high-quality movements.

In the following we will see, that the imitation learning algorithm proposed can be used to overcome above mentioned problems. The steps involved in this process are depicted in Figure 6.1. Data collected from a full-body motion capture session is used to learn a compact model of the observed behavior using the PLDPM construction algorithm.

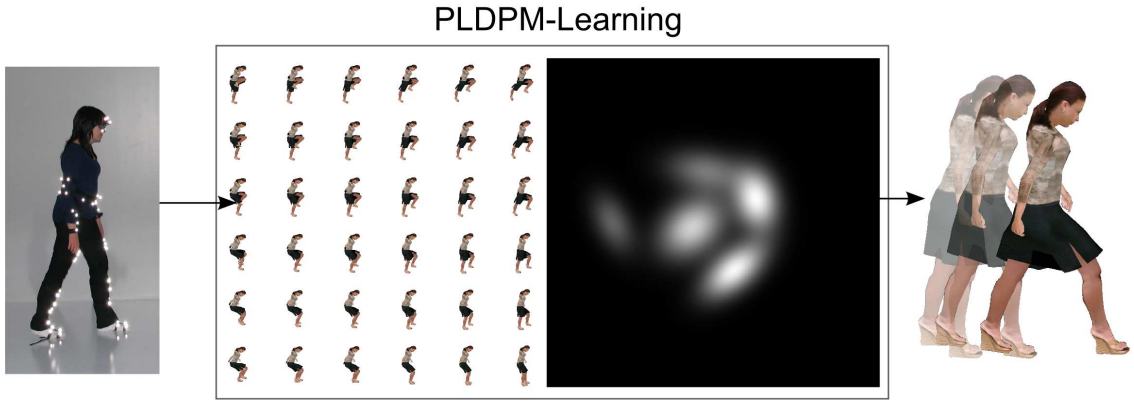


Figure 6.1: Imitation learning of a walking skill from motion capture data. After recording the data, the PLDPM construction algorithm is used to produce a compact model of the behavior. The model can then be used to synthesize new walking gaits with respect to the current terrain.

In a similar way as in our previous example, we can use the extracted posture space and probability density function to generate a variety of walking gaits. At runtime, the PLDPM is used in conjunction with optimization techniques to synthesize a new walking gait which best fits the current terrain as well as the anatomy of the virtual human. By the combination of motion capture data, imitation learning and optimization, we can generate adaptive, high-quality animations.

We will demonstrate the method based on two behaviors: *'walking'* and *'climbing up a stair'*. The goal is to have virtual humans that can step over gaps or adapt the height of their step to the level of a staircase. At the same time, we will see, that the PLDPM approach allows for intuitive visual inspection and analysis of a behavior. Note that, although the following discussion focuses on locomotion skills, the approach is also applicable to any other kind of physical motor skill.

6.2.1. Kinematic Modeling of Virtual Humans

Virtual humans are articulated by skeletal modeling techniques, as used for the hand example in Chapter 5. The configuration of the kinematic chain representing the skeleton of a virtual human is modified to produce different postures thereof. The mesh of the virtual human is automatically deformed to fit the new skeletal configuration. In Figure 6.2, we see a visual representation of a skeleton, as well as the resulting pose. The figure also includes the names of joints which are relevant to the subsequent experiments. The joints are named according to the naming convention of the H-ANIM standard [H-A10],

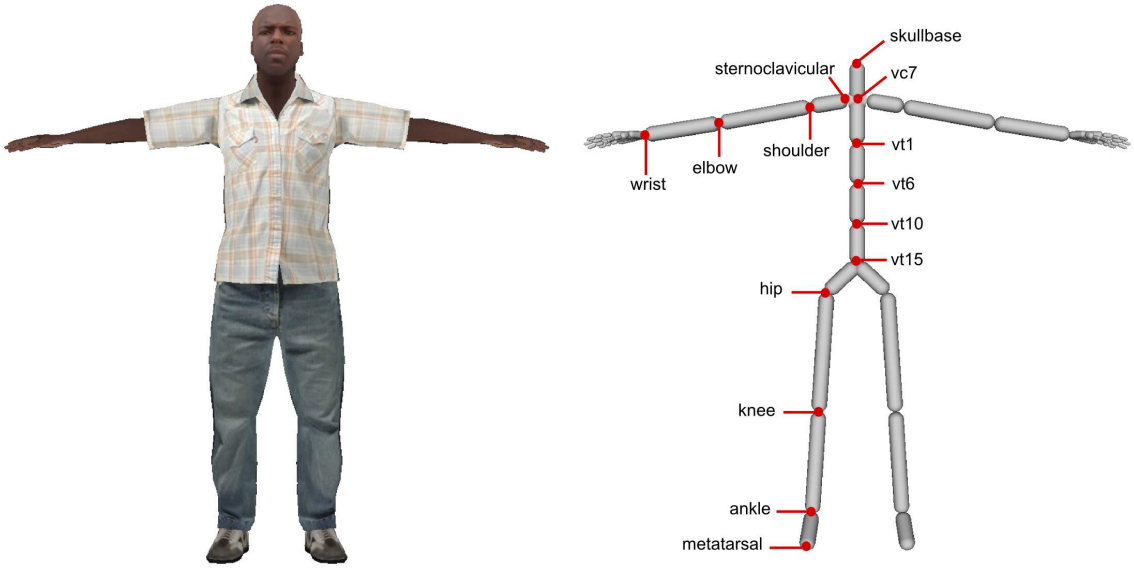


Figure 6.2: Joint names according to the H-ANIM standard.

which defines a common denominator for names, orientations and hierarchies of bones used for animating virtual humans.

In order to articulate a virtual human and produce realistic animations, the state of the kinematic chain must be changed for every frame. The joint orientations of all modeled joints need to be provided for each frame. Such an animation scheme controls characters on a purely kinematic level; no physics is used. The inclusion of physics will be discussed in Section 6.3.

Similarly to real humans, their virtual counterparts come in all sorts of shapes and sizes. Figure 6.3 shows a set of virtual humans with varying anatomical properties, e.g. tall and small characters. These differences need to considered when synthesizing walking gaits. Applying the same joint orientations on a small and a tall virtual character will likely result in different step sizes. Due to the length of the limbs a tall character can make larger steps. If we want to ensure that both characters walk with the same speed, we need to generate more steps for the smaller character. This problem is further intensified by the fact that virtual environments allow for arbitrary anatomical configurations, including exaggerated proportions of the limbs of cartoon or alien characters.

6.2.2. Motion Recording

The training data was recorded using a motion capture system by ART [ARTG10]. A set of twelve cameras with a frame rate of 60Hz was used for tracking. Six students, three men and three women, served as models for collecting example locomotion data.

Figure 6.3: Virtual humans with varying anatomical properties.

Their ages ranged between 18 and 23 years. To acquire the motion data, each participant was equipped with a set of 59 retroreflective markers. The markers were attached to 12 different targets used to track the motion of the head, arms, pelvis, legs and feet of the participants.

For recording 'walking' examples, the participants were asked to walk in normal speed inside the tracking spot of the ART cameras. For 'stair climbing', a simple setup of plastic boxes was used to simulate a staircase. The participants were asked to climb up this setup as if it were a staircase. During each motion capture session, the ART cameras recorded the position of the markers and forwarded them to a tracking server. The server then analyzed the recordings and calculated global orientations of the targets. In turn, our animation system used these orientations to derive twelve local joint orientations for the kinematic chain of a virtual human.

The recorded training data was then processed using the PLDPM construction algorithm (see Chapter 4). Table 6.1 shows the reprojection error for different dimensionality reduction techniques and different dimensions. Please note that the reprojection error is computed by calculating the difference between the original data points and the reprojections. For a posture space with two dimensions, PCA introduces an error of about 30° for each posture. Divided by the number of joints, this translates to an error of about 2.5° per joint. Analyzing the results of Table 6.1 in more detail, we find that overall PCA, MDS, and NLM outperform other techniques for this dataset. This is quite astonishing, given the fact that these methods are among the first dimensionality reduction techniques to be invented. Further, two of these three techniques, namely PCA and MDS, are linear methods, which are considered to be less powerful than their nonlinear counterparts.

Figure 6.4 depicts the first three principal components of the walking dataset, as computed with PCA. Moving along the first principal component, the legs of the virtual human are bent to perform forward steps. In the negative direction of this axis the left

Dim.	PCA/°	LLE/°	CCA/°	CDA/°	MDS/°	ISOMAP/°	NLM/°
1	38.9772	46.1369	55.5424	43.6217	38.2014	41.2223	37.142
2	32.7846	36.912	45.5224	74.5078	31.5238	34.2774	32.8997
3	31.2088	36.8654	66.7887	69.2383	29.8778	32.266	29.3771
4	27.7332	37.8122	70.0059	77.0528	26.9743	29.7088	27.9316
5	26.0983	36.7741	76.9623	77.0878	25.1738	30.4915	25.239
6	24.1097	35.0048	97.6533	85.065	22.0688	29.8557	22.3293
7	21.2586	32.8904	75.3553	60.7615	20.7132	28.3085	21.4665
8	19.4655	31.9327	78.7092	79.2587	19.7922	26.5732	20.0625
9	16.6463	27.5094	79.6683	94.6961	18.34	25.856	19.1346
10	25.3733	36.8526	74.5771	77.1858	27.1986	31.2925	25.6252
Ø	26.3656	35.8691	72.0785	73.8476	25.9864	30.9852	26.1208

Table 6.1: Reprojection error in [degree] for reconstructed 'walking' postures.

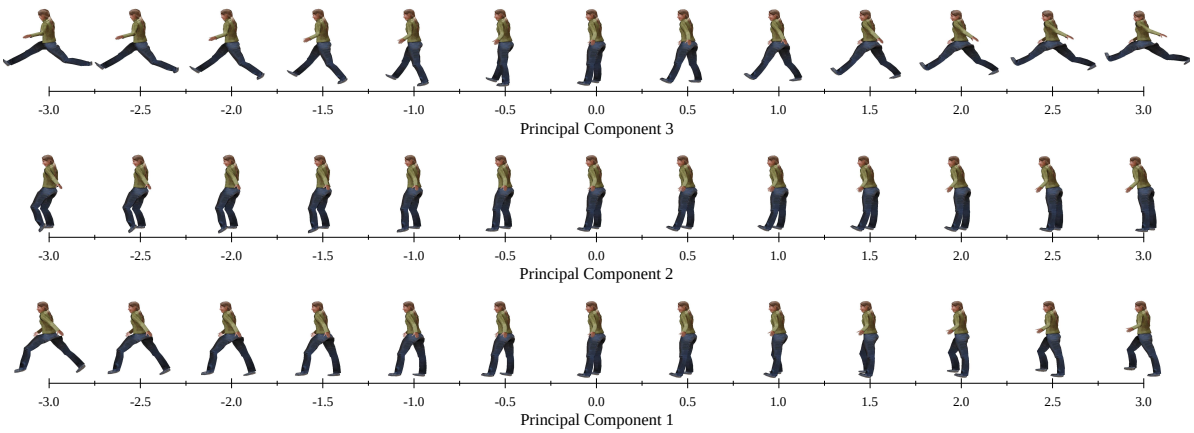


Figure 6.4: Visualization of the first three principal components of walking gaits computed by PCA.

leg is moved forward, while on the positive part the right leg is moved forward. The second principal component corresponds to different balancing poses. In the positive direction of the axis the virtual human leans backwards, while in the negative part it leans forwards. Finally, the third principal component corresponds to taking steps of different sizes.

As in our earlier examples, this shows the ability of PCA to extract meaningful components. Each of the above components reflects a property of walking gaits, which can also be described in simple terms. More interesting discoveries can be made if we analyze the

space spanned by the first and the third principal component. A visual representation of this space can be seen in Figure 6.5. Equidistant points in the space are reprojected and the corresponding posture of the virtual human is displayed. The result is a whole continuum of postures, which are typical for walking. The origin of this space corresponds to a standing posture, the legs are not extended at all. The more we move to the corners of the space, the more *extreme* the postures of the virtual human.

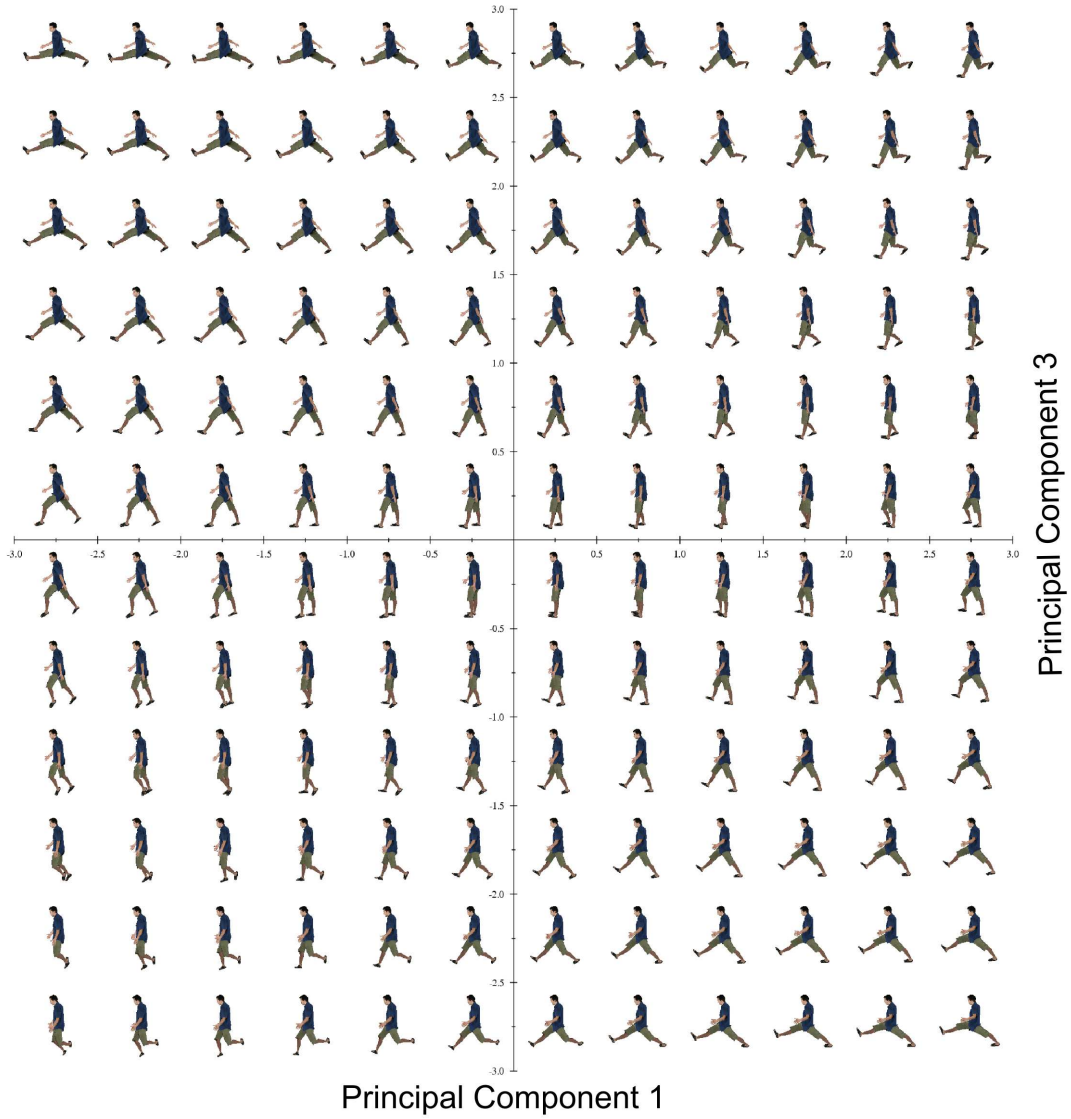


Figure 6.5: Visualization of the posture space spanned between the first and the third principal component.

The legs become more and more stretched and are more reminiscent of running than walking. We also observe that nearby points in the above space correspond to similar postures of the virtual human. In other words, the space is continuously differentiable. Small changes introduced to a low-dimensional point in this space, lead to small changes in the reprojected kinematic configuration. As already discussed in Chapter 4, this property of the posture space allows us to generate smooth animations by specifying continuous trajectories.

The most interesting observation, however, is made by if we analyzing the postures along a circular trajectory inside the posture space: every such circular trajectory (centered at the origin) corresponds to a walking gait with two steps. Circles with small radius represent slow walking, while circles with larger radius represent faster walking, or even running. To confirm this finding, we projected recorded walking motions of participants into the posture space. Figure 6.6 shows a typical trajectory resulting from such an operation.

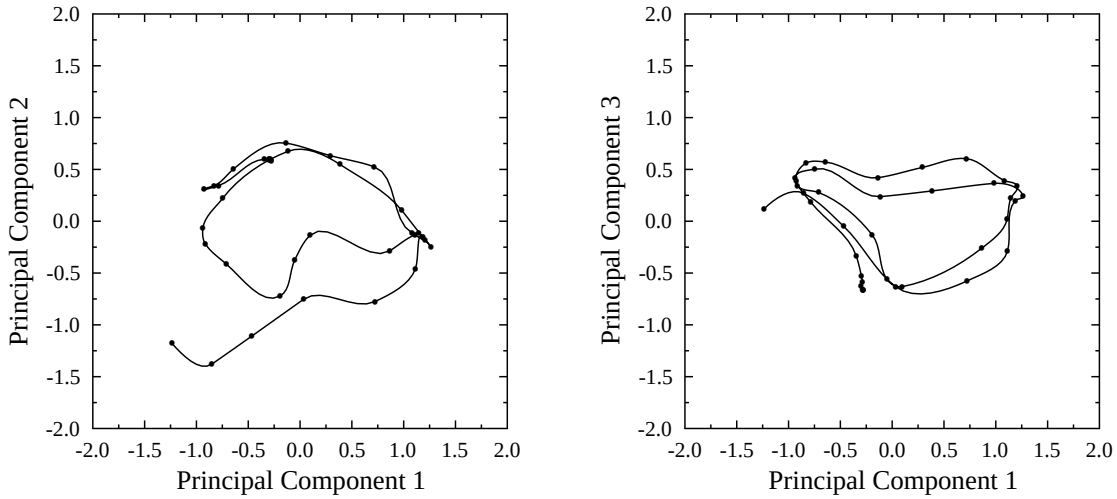


Figure 6.6: A low-dimensional trajectory resulting from projecting recorded walking motion into the posture space.

While the trajectory does not correspond to a well-shaped circle, it still features a roughly elliptic, circular shape. Because the participant performed several steps, the trajectory contains two overlayed circular shapes. To further undermine our hypothesis, that circles in the posture space correspond to different walking gaits, we next analyze the change of step size. In a normal walking gait, the distance between the feet increases when the left leg is moved forward. In the next phase, the right leg is moved forward until it passes the left leg. In this process, the distance between the feet decreases and

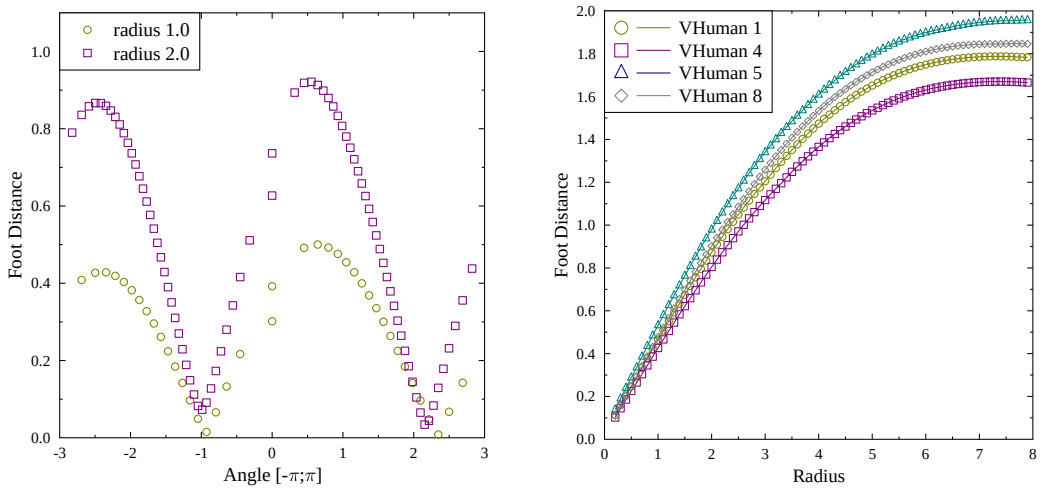


Figure 6.7: Analysis of the distance of the feet of the virtual human. Left: Two circles with different radii are specified in the low-dimensional space. A radius and an angle (x-axis) specify a point in polar coordinates in the low-dimensional space. The y-axis holds the distance of the feet of the corresponding posture. Right: The maximal distance of the feet for different virtual humans and different radii of the low-dimensional circles.

increases again. In other words, the distance of the feet in a walking cycle obeys a rhythmic up and down movement, similar to a sine wave. To check if this also translates to animations generated from circular trajectories in posture space, the following test was performed. Circles with different radii were specified in the space spanned by the first and second principal component, and the postures along these trajectories were computed through reprojections. Then, for each of these postures, the distance between the feet of the virtual human was computed. Figure 6.7 (left) shows the resulting distances. The positions in the posture space were encoded in polar coordinates. The abscissa of the graph corresponds to the angular value, while the different colors and markers encode the radius.

The graphs contain a sinusoidal, repetitive oscillation with two peaks. The peaks correspond to the two phases of maximally extended limbs; one peak for the left leg, and one for the right leg is moved forward. Based on the above results, we can also confirm that the radius of a circle determines the speed of the walking gait. For example, a radius of 2.0 in the posture space translates to a maximal distance of the feet of about 90 cm, while a radius of 3.9 generates a maximal distance of about 1.5 m. In order to understand the relationship between the circle radius and the maximal feet distance, an additional experiment was performed. Here, the radius of the circle was slowly increased and the maximal distance during the animation was recorded. The experiment was performed for different virtual humans, in order to reveal the influence

of the anatomy on the maximal stepsize. Figure 6.7 (right) shows the maximal stepsize as a function of the circle radius. The differently colored curves show how the values change for different virtual humans. The anatomy of a virtual human obviously has an effect on the maximal step size. To summarize our observations, we can say that walking gaits can be synthesized by generating cyclic trajectories—in the most simple case, circles—in posture space. The size of the circle has to be optimized to generate goal-directed behavior. This optimization takes into account both the anatomy of the virtual human, as well as the current environment and the intended destination.

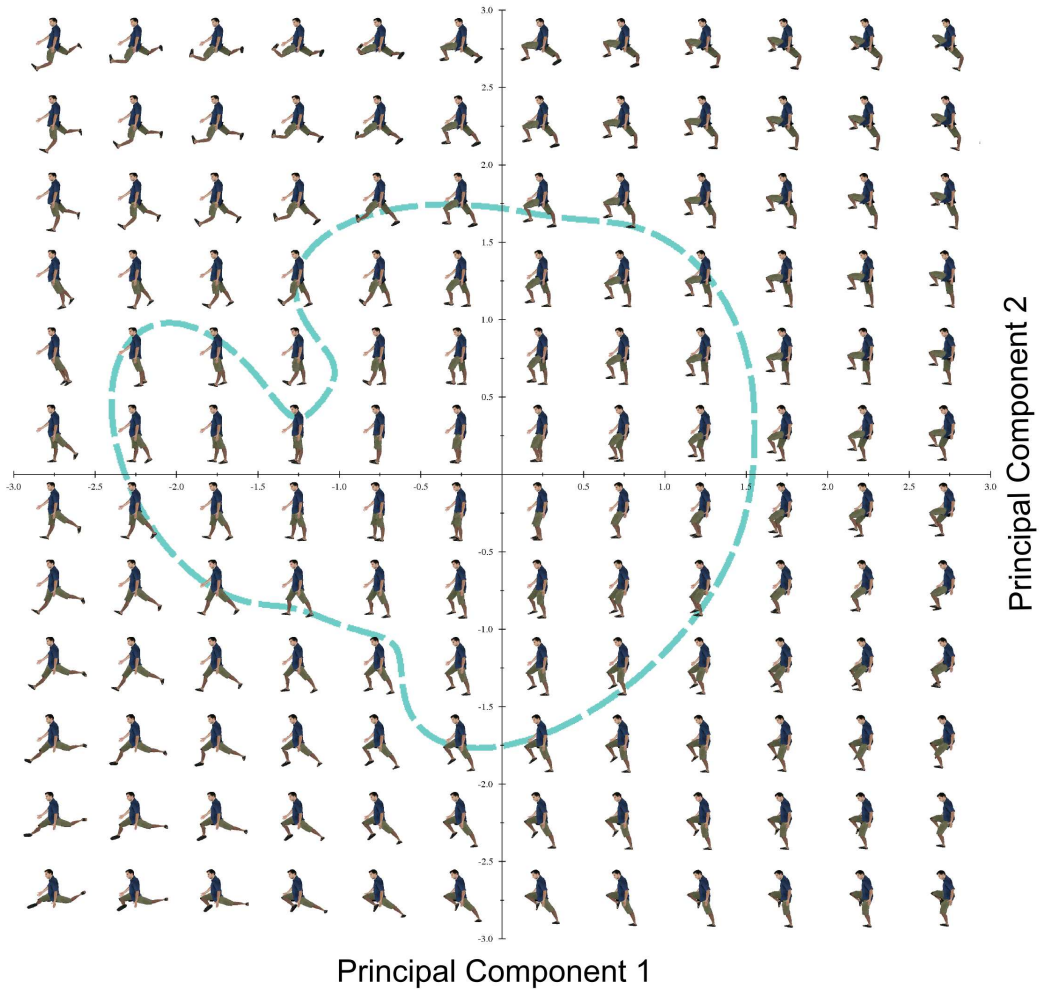


Figure 6.8: Visualization of the posture space generated from stair climbing data. The dashed blue line shows the area of high probability. According to the probability density function learned from the data set, postures inside this area are likely to be anatomically feasible.

Furthermore, we analyzed the PLDPMs generated for the stair climbing data sets, Figure 6.8 shows the corresponding posture space. In addition to the postures, we also plotted the border of the high-probability area inside the posture space. According to the probability density function learned from the data set, all postures inside this area are likely to be anatomically plausible. In contrast to the walking example, stair climbing does not correspond to a circular trajectory in posture space. Instead, more complex trajectories must be specified to generate an accurate stair climbing animation. The shape of the trajectory strongly depends on the dimensions of the staircase elements. During optimization we must, therefore, ensure that the synthesized animation perfectly matches the height and depth of the elements. Otherwise, the legs and feet of the virtual human would penetrate the staircase.

6.2.3. Motion Synthesis

The metric of imitation of both the 'stair climbing' and the 'walking' behavior is inspired by dancing instructions, i.e. the way dance steps are documented. Written dancing instructions often include a diagram visualizing the involved moves as a sequence of footprints. Such diagrams convey information about timing, location and side of the dancing step. Figure 6.9 (left) illustrates how this concept can be applied to our domain. A set of footprints for the left and right foot is specified relative to the depicted staircase. The footprints can be seen as constraints that must be met in order to generate a successfull stair climbing animation. Contact between the virtual human's feet and the ground must be ensured at each of the specified footprint locations. The footprints show the alternation of left and right foot that is typical for locomotion. Note also, that these footprints are specified relative to the shape of the ground. As a result, regardless of how the size of the staircase or the characteristics of individual stairs changes, the footprint will be correctly placed on top. The same principle, of course, also applies to walking gaits or other locomotion tasks.

Given these footprint constraints, how can we synthesize an animation? As documented by the principle of posture-point duality (see Section 4.3), the synthesis of a new animation involves the specification of a trajectory in the low-dimensional posture space. This trajectory is based on a set of control points, each corresponding to a posture fulfilling one of the footprint constraints. In other words, for each constraint, we need to specify a corresponding point in posture space. The trajectory connecting all of these points then represents the target animation. But how to find for each of the constraints a corresponding low-dimensional point, which meets the requirements?

We will use a similar approach as in the *'button pressing'* example in Section 4.4. For each constraint, we search the posture in the posture space, which minimizes the distance between the sole of a foot and the respective footprint position [see Figure 6.9 (middle)]. Additionally, the difference between the orientation of the foot and the footprint can also be added to the fitness function. The goal of optimization is, hence, to find a set of points in the posture space that minimizes the deviation in position (and possibly

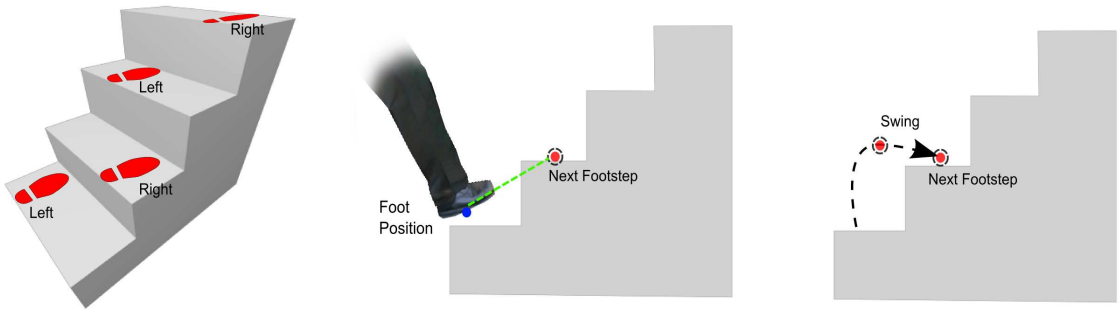


Figure 6.9: Calculation of the fitness for locomotion behaviors. Left: Desired footprint positions are defined based on the structure of the terrain. Middle: The distance of the foot to the specified footprint is minimized. Right: To simulate a more realisitic leg motion, intermediate footprints can be specified.

orientation) between the leg of the virtual human and the footprints. The number of control points for the low-dimensional trajectory is equal to the total number of stairs in the staircase.

However, in order to improve the visual quality of the synthesized animation, we can also specify additional constraints and consequently control points than the number of stairs. For example, we can add a constraint for the position of the foot during the swing phase of the motion, as can be seen in Figure 6.9 (right). By specifying such intermediate positions through which the foot has to pass, we can control the transition of the feet from one step to the other in more detail. This can be used to enhance the dynamic character of the resulting animation. In addition, such intermediate positions can be used to avoid penetrations, which would normally occur, if a foot is moved on a straight line from one step to the other.

To evaluate the suitability of the fitness function described, we conducted a series of experiments. In these experiments, we synthesized stair climbing and walking animations using different virtual humans. We then analyzed the results both in quantitative (foot deviation) and qualitative terms (animation quality). Table 6.2 shows the results of the quantitative analysis for the 'stair climbing' behavior. The experiment was conducted with PLDPMs constructed using different dimensionality reduction techniques, and with varying dimensionality of the posture space. For each combination we measured the distance between the desired position of the foot (the footprint location) and the synthesized position of the foot. The results in Table 6.2 correspond to the mean values of these distances (in meter). We can see that with a posture space of four dimensions the discrepancy is already in the range of millimeters for most of the DR techniques. Again, the best results are achieved with PCA, MDS and NLM with deviations of one to two millimeters.

Dim.	PCA/m	LLE/m	CCA/m	CDA/m	MDS/m	ISOMAP/m	NLM/m
2	0.37	0.055	0.052	0.023	0.039	0.033	0.046
3	0.19	0.022	0.022	0.0065	0.0012	0.0097	0.0088
4	0.0012	0.0104	0.012	0.0020	0.0019	0.01	0.0026
5	0.0018	0.0122	0.0037	0.0033	0.018	0.001	0.0010
6	0.0038	0.0009	0.0004	0.0008	0.0029	0.019	0.0016

Table 6.2: Per-step optimization error in [meter] for the 'stair climbing' behavior.

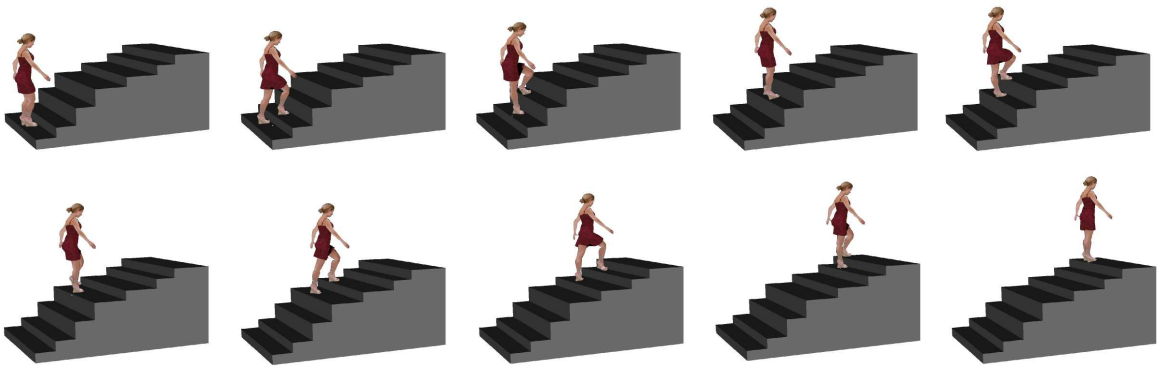


Figure 6.10: A synthesized stair climbing animation. The movements are adapted to the different heights of the stair.

Judging from these results, we can conclude that four dimensions are sufficient for generating walking and stair climbing animations that closely follow a specified sequence of footprints. A discrepancy of one to two millimeters is generally not perceivable by a human user. Additionally, we analyzed how well the approach can adapt to extreme variations of the environment. We modeled a virtual staircase with stairs that vary considerably in height and depth. The staircase consisted of stairs with a height of 15 cm or 30 cm, and a depth of 20 cm, 30 cm, or 40 cm. In order to produce viable animations, our synthesis algorithm must correctly adapt the gait of the virtual human to the dimensions of the stairs. Figure 6.10 shows pictures from a synthesized stair climbing animation. We can see that the posture of the virtual human adapts to the height and depth of the current stair. In particular at the second stair (third picture in the upper row), we can see that the virtual human makes a larger step in order to successfully place the foot on top of the higher stair. In this situation the virtual human lifts the leg up in a more vertical motion. On the smaller and deeper steps, however, the virtual human moves the leg less upwards and more forwards. More generally, the

figure shows that our imitation learning algorithm is well suited for synthesizing new animations which are well adapted to the current environmental context.

Finally, we also evaluated the visual quality of the walking gaits produced by our algorithm. Realistic human walking gaits show a specific alternation in swinging the legs and arms. Each arm moves forward in synchrony with the corresponding leg on the opposite side. From a biomechanical point of view, it increases the power efficiency of the walk. From a visual point of view, the swinging of the arms increases the naturalness of a walking gait. However, in our approach we do not explicitly model this type of dependency between different body parts during walking. Instead, this task is left to the PLDPM construction algorithm, which extracts this information from the training data. So although the fitness function does not address the relationship between the arms and the legs, we still expect correct oppositions of arms and legs in the synthesized animations. Figure 6.11 shows a sequence of pictures from such a synthesized walking animation.

Figure 6.11: A walking gait synthesized by our imitation learning approach. Although the animation is generated through optimization it appears highly natural. In contrast to traditional motion capture, we can adapt the animation to any terrain or desired footprint location.

As for the stair climbing example, the discrepancy between the desired leg position and the generated leg position in the synthesized walking animations is within a few millimeters and therefore not perceivable for the human eye. As indicated by the postures shown in Figure 6.11, the movement appears natural and is clearly recognizable as a walking gait. The arms also swing in synchrony with the opposite legs without this being explicitly specified in the fitness function. This leaves us with a method for adaptively synthesizing new walking gaits with a realistic appearance. The footprint locations can be modified according to the underlying terrain or the intended speed of locomotion. The synthesis algorithm ensures that a walking gait is produced which passes through these footprints. At the same time, new models can be trained in order to synthesize walking gaits with a different style.

6.3. Programming Robots by Demonstration

Moving from virtual humans to other types of synthetic humanoids, we will subsequently show how small humanoid robots can be trained using a variation of the proposed imitation learning approach. More specifically, we will present a new programming by demonstration method for bootstrapping robotic motor skills through kinesthetic interactions.

Figure 6.12: A human teaches a small humanoid robot how to stand up.

In this method, a human teacher instructs a robot by manually moving the robot's joints and body to postures that approximate the intended movement. Figure 6.12 depicts such a situation, where a human teacher showing a small humanoid robot how to stand up. After instruction, an automatic optimization phase takes place during which the robot learns a motor skill that still resembles but also compensates for likely imperfections of the demonstrated movement. This learning phase makes use of a *physics-based virtual environment*, where a large amount of movement variations can be tried out very quickly without the need for human intervention. Henceforth this new imitation learning approach will be called *kinesthetic bootstrapping*.

6.3.1. Kinesthetic Bootstrapping

When learning a new physical skill, children are often supported by their parents. This allows to transmit knowledge on how to solve the task at hand and, thus, overcome learning barriers. In these situations, kinesthetic interactions serve as a communication channel between the parent and the learning child. The bodily experience resulting from these interactions helps to reduce the amount of time needed for acquiring the skill. Still, the child has to go through an unassisted learning phase in order to fully master the skill. Kinesthetic bootstrapping applies the same principle to the programming of humanoid robots. A human conveys a demonstration of the task at hand through kinesthetic interactions. The bodily experience allows the robot to draw important information on the task at hand. This information is used to "bootstrap" the robot's knowledge, which is then used in a learning phase to reproduce the skill without any assistance. Figure 6.13 shows an overview of the learning approach used in kinesthetic bootstrapping.

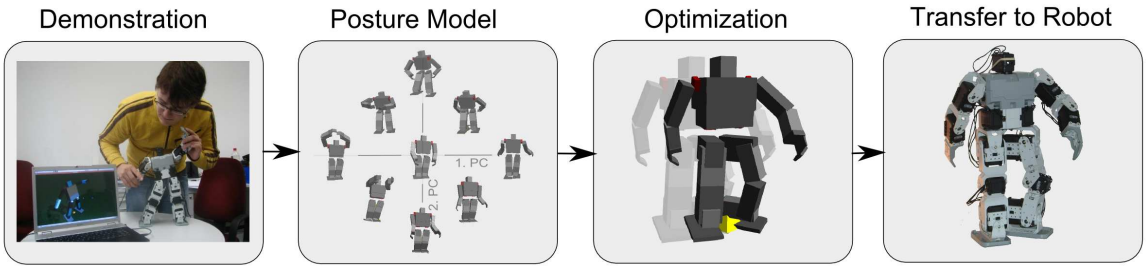


Figure 6.13: Overview of the kinesthetic bootstrapping approach. After kinesthetic interaction, a posture model is created. A simulator is used to optimize the demonstrated skill. The result is then applied on the physical robot.

First, the teacher moves the joints of the robot in order to convey a demonstration of the intended motion or behavior. This is done continuously without relying on keyframes or another kind of discretization. During the demonstration, the motor configurations of the robot are recorded with a frequency of 20 Hz in our case. The robot used in this study is a *Bioloid* robot with 18 servo-motors (i.e. 18 DOF). In each step, the state of each of the servo-motors is recorded, resulting in an 18-dimensional posture vector $\boldsymbol{q}$. Once the user finishes the demonstration, all posture vectors are collected in the set $\mathcal{Q}$ in order to compute a low-dimensional posture model of the skill. Next, using the extracted posture model, different variations of the skill are evaluated. This is done in a physics-based virtual reality simulation of the robot. The simulator allows us to optimize the motion without harming the robot hardware and without relying on human assistance. In particular, when the demonstrated motion is very dynamic, such as a standing up motion, it is important for the robot to learn how to account for the external (stabilizing) forces previously applied by the human teacher. We can also see in the figure that kinesthetic bootstrapping extends our typical three-step approach to imitation learning by a fourth step, namely a step for transferring the behavior to the robot. Of course, it is not sufficient to synthesize and replay the robot motion in simulation only. Once the optimization phase is finished, the learned motion must be transferred to the physical robot and replayed outside of simulation.

The simulator used for kinesthetic bootstrapping is based on the Open Dynamics Engine (ODE) and contains a precise model of the Bioloid humanoid robot. For calibrating the model, each motor is automatically moved by the calibration software and the time needed to reach a given configuration by the real and simulated robot is measured. The difference between the two time values, i.e. the discrepancy between the real and simulated world, is used to adapt the values of the low-level PID controller in simulation, so as to better fit the movements of the real robot. An important feature of this simulator is an abstraction layer for the control of the robot. This layer allows us to control the real or simulated robot or both using the same interface. Thus, the user can

always decide whether to apply the current program in reality or simulation. During the learning phase, the simulator is used for reproduction of different variations of the originally demonstrated motion. In a trial-and-error fashion each variation is executed by the virtual robot, evaluated and the result is used for further optimization. In the remainder of this section, we will explain each phase of the learning process in more detail.

6.3.2. Learning

The kinesthetically recorded demonstration can be regarded as a template from which important information about the skill in mind is inferred. Following our typical imitation learning approach, we can therefore learn a PLDPM from dataset $\mathcal{Q}$. The resulting low-dimensional space of postures can have arbitrary dimensions L, with $L \ll 18$. Without loss of generality, in the following explanation, we will use a two-dimensional posture space ($L = 2$).

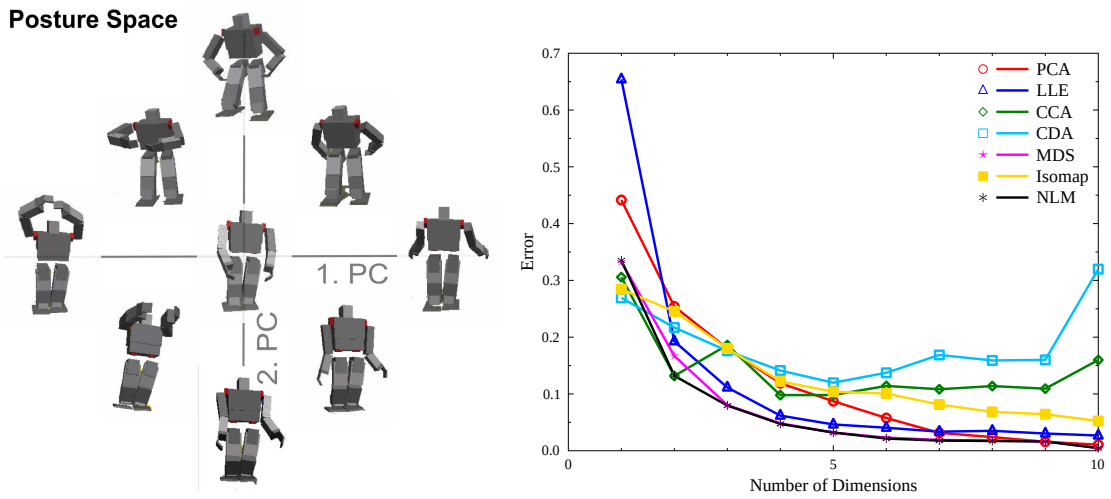


Figure 6.14: Left: A representation of the low-dimensional space corresponding to the posture model for grasping with both hands. Right: The projection error resulting from applying different dimensionality reduction techniques to the recorded robot postures.

In Figure 6.14 (left) we see an example of a learned posture space for the Bioloid robot. The model was computed based on demonstrations of two-handed grabbing or grasping. Figure 6.14 (right) depicts the reprojection error generated by applying different dimensionality reduction techniques to the data set of postures. We found that for most skills, even with a simple PCA, 95% of the original information can be retained using only four principal components. As in our previous examples, full-body motions for the robot can be synthesized by simply specifying a trajectory in the posture space.

Each point along the trajectory reflects a posture of the robot at a particular time step of the motion.

After learning a PLDPM, an evolutionary algorithm is used to optimize the demonstrated skill. As a starting point for optimization, we use the low-dimensional trajectory of the demonstration. For this, the set $\mathcal{Q}$ is projected into the posture space, yielding a new set $\widetilde{\mathcal{Q}}$ of points specifying a L-dimensional trajectory. Next, $\widetilde{\mathcal{Q}}$ is approximated using n control points $\mathcal{S} = \{\boldsymbol{s}(1), .., \boldsymbol{s}(n)\}$ specifying a spline curve. The spline can be regarded as a highly compressed representation of the demonstrated motion. Instead of using all points of the original trajectory $\widetilde{\mathcal{Q}}$, only a limited number n of control points is used for optimization.

More precisely, the control points $\mathcal{S}$ are used for initializing individuals of an evolution strategy. A set of slightly perturbed variants of $\mathcal{S}$ are created in the initial population of the ES. Each individual is then processed, and the corresponding motion executed by the simulated robot. This is done by reprojecting each point along the encoded trajectory back to the original space of joint values. Using a user-provided fitness function, each individual is then evaluated and assigned a fitness value. Once all fitness values are determined, the best individuals are selected, mated and mutated according to the typical rules of a ES (see Section 4.4.3). Finally, when the learning process is finished, the newly learned skill is applied on the real physical robot.

When performing dynamic motions, such as a standing up behavior, timing plays an important role. Therefore, we add a special time parameter for each of the control points into the chromosome. The time parameter indicates at which timestep each posture should be realized. Each individual in the ES, thus, consists of a set of values $\{\boldsymbol{s}(1), t(1), .., \boldsymbol{s}(n), t(n)\}$.

6.3.3. Experiment and Results

To evaluate the proposed approach, we conducted a set of experiment in which a human teacher had to teach a small humanoid robot a set of skills using kinesthetic bootstrapping. Among others, the robot learned to stand up by itself, walk, and perform a headstand. In all experiments, PCA was used as a dimensionality reduction technique. The number of dimensions L was set to 4. In contrast to our previous examples, optimization was performed cast as a maximization (instead of minimization) problem. That is, the higher the fitness values the better.

The teacher was given about 15 minutes time to kinesthetically demonstrate the respective skill. In the upper row of Figure 6.15 we see the result of directly replaying the demonstrated skill. Because of the missing support of the teacher, the robot failed to stand up by itself. In order to successfully stand up, the robot must learn to compensate for these external forces using his own motors. For this, we ran the optimization as described in Section 6.3.2. The number of control points n in a trajectory was 25. For the 'standing up' skill, the fitness values were determined based on the sum of the z-values (=height) of the robot's head position. That is, the higher the robot rose up,

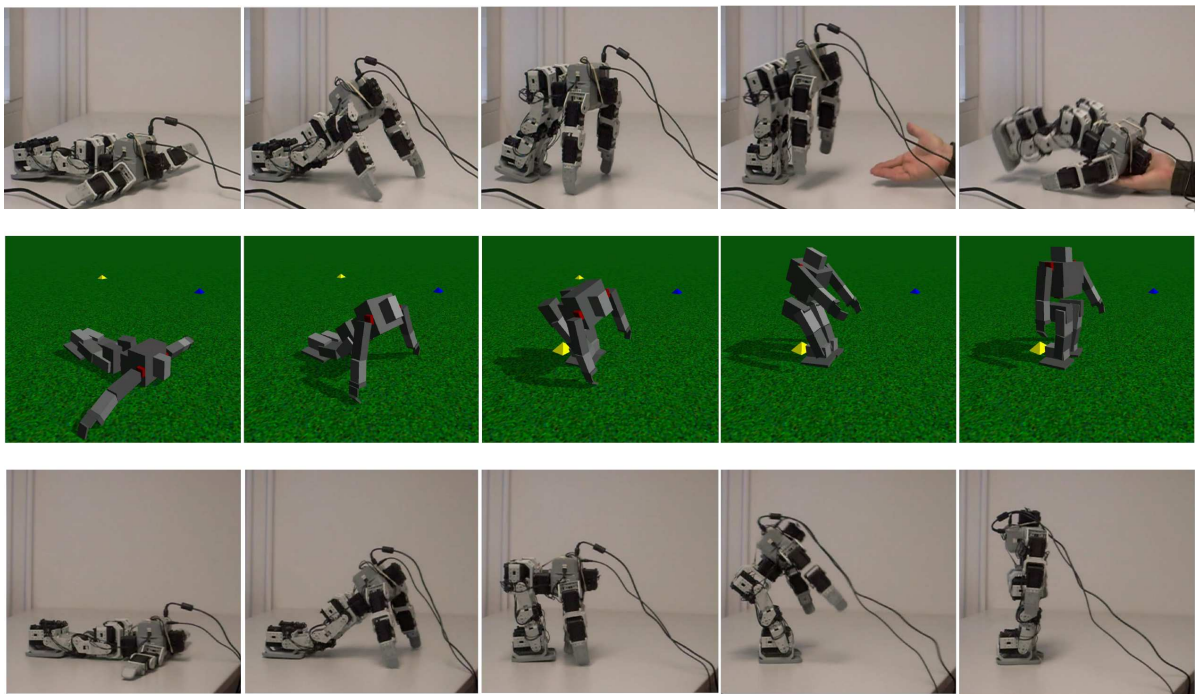


Figure 6.15: Upper row: Direct replay of a demonstrated standing up skill by the small humanoid robot. The robot fails to stand up, because of the missing support forces of the human teacher. Middle and bottom row: Results of applying the evolved standing up behavior in simulation and on the real robot. The robot learned to move the hip backwards to an extreme position, so as to pull up the torso without falling forwards.

the higher was its fitness. The trial was aborted, if the robot's head was below a given threshold, i.e. the robot fell down during the simulation. The middle and bottom row of Figure 6.15 show the result of the optimized skill in simulation, and after application on the real robot. As can be seen, the robot learned to stand up by modifying the original motion. In particular, the hip motion was changed such that the robot can lift its torso without losing balance (second picture from right). By moving the hip backwards to an extreme position, the center of mass of the robot remains between the legs. The robot, then, can rapidly pull up his hands without falling over. As can be seen in the figure, the execution in both the simulator and on the real robot leads to approximately the same motion. In both cases, the result is an elegant solution to the problem of standing up.

For the walking skill, the fitness value of each individual was determined, using the distance traveled from the starting position without falling down. The robot received a high fitness if it successfully travelled a long distance. Note, that this is a more abstract form of objective function than the one used for locomotion in virtual humans. In the case of the virtual humans, we explicitly specified the positions where the feet of the

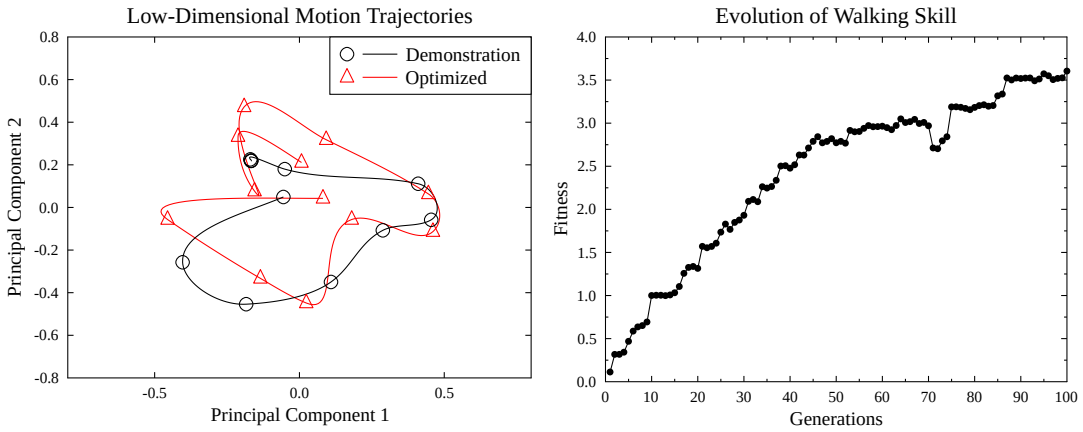


Figure 6.16: Left: The trajectories of the walking skill in the low-dimensional space of postures. Right: The evolution of the fitness value during the learning of the skill. The fitness was determined using the distance traveled by the robot.

virtual humans should be, as well as the timing in which this should occur. In contrast to that, in this example we do not provide the evolutionary algorithm with this information. The only information on which optimization is based, is the distance travelled during the execution of a particular individual of the evolutionary algorithm. The number n of control points was set to 12. In Figure 6.16 (left) we see the low-dimensional trajectories resulting from the control points of the walking skill before and after optimization. The trajectories show the values of the control points in the first and second principal component. The lower-order components, in this case the first and second component, contain the 'most important' aspects of the data. Thus, by visualizing the first two components, we can see most important changes to the robot motion. We can see in the figure that some control points of the trajectory were only slightly modified, while in other parts strong modifications were made. The resulting optimized trajectory resembles a skewed version of the original trajectory. Figure 6.16 (right) shows the evolution of the fitness values during optimization. In the first generation the fitness value is near zero because most of the synthesized walking gaits resulted in the robot falling down. However, as optimization progressed, the fitness gradually improved.

With each generation of the ES, the robot managed to travel larger and larger distances. However, after the ES finished, we found that the best individual in the simulation did not lead to a stable walk in the physical system (real robot). This unveils a common pitfall of using a simulator: even the best simulation is only an approximation of the real world. Fortunately, ES allows for a simple solution to this problem. By testing the best individuals of earlier generations, we can search for solutions that are transferable to the real world. Figure 6.17 shows an evolved stable walking pattern. It corresponds to the fittest individual from generation 50. In later generations, the ES ex-

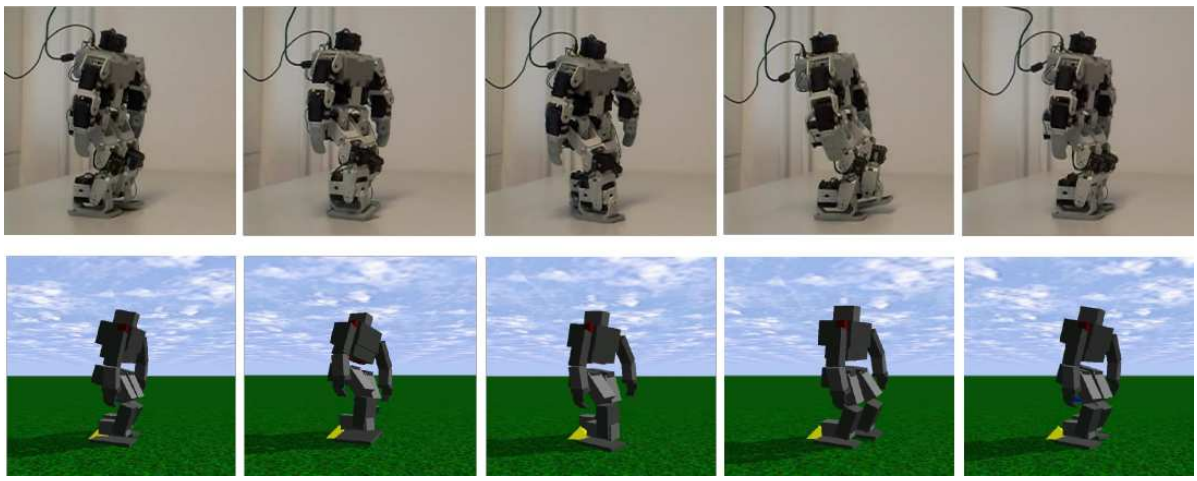


Figure 6.17: The result of applying an evolved walking behavior in simulation and on the real robot. Learning this skill involved a kinesthetic demonstration of only 5 minutes, in which the legs were moved by the human, and the specification of the fitness function.

ploited the characteristics of the simulator too much and, thus, generated an individual that was not applicable to the real environment.

The *'headstand'* behavior is a particularly difficult skill for a robot to perform. Indeed, even for humans this skill can be difficult to execute, as it requires the right amount of force, balance and timing. To successfully perform a headstand, the robot first needs to move its arms in a triangular shape in front of its torso. Next, the pelvis must be moved in direction of the head and the legs need to be straightened. Finally, the legs must be lifted up and the weight needs to be correctly balanced, i.e. the robot should not fall sideways. The whole sequence has to be performed dynamically in order to generate enough force and keep balance. The headstand is optimally executed if the robot can stretch out the whole body while keeping balance on the head and the support built up by the arms.

The upper row of Figure 6.18 depicts the result of directly replaying a demonstrated headstand behavior. The robot cannot perform this skill successfully and slips on the surface of the table. The main reason for this problem, is that the robot prematurely lifts up the legs. Without the supporting forces of the human demonstrator, the robot's legs fall back onto the surface of the table where they generate friction. In turn, this friction results in the torso being moved forwards, instead of the legs being lifted up. To overcome this problem, the robot must adapt the demonstrated motion to its own dynamics and build up sufficient momentum and balance. As in the previous examples, this was done in optimization. As a fitness function, we used the sum of the z-values of the robot's left and right foot. The more the robot succeeded in pulling the legs and feet upwards, the better was his fitness. As can be seen in the bottom row of Figure 6.18,

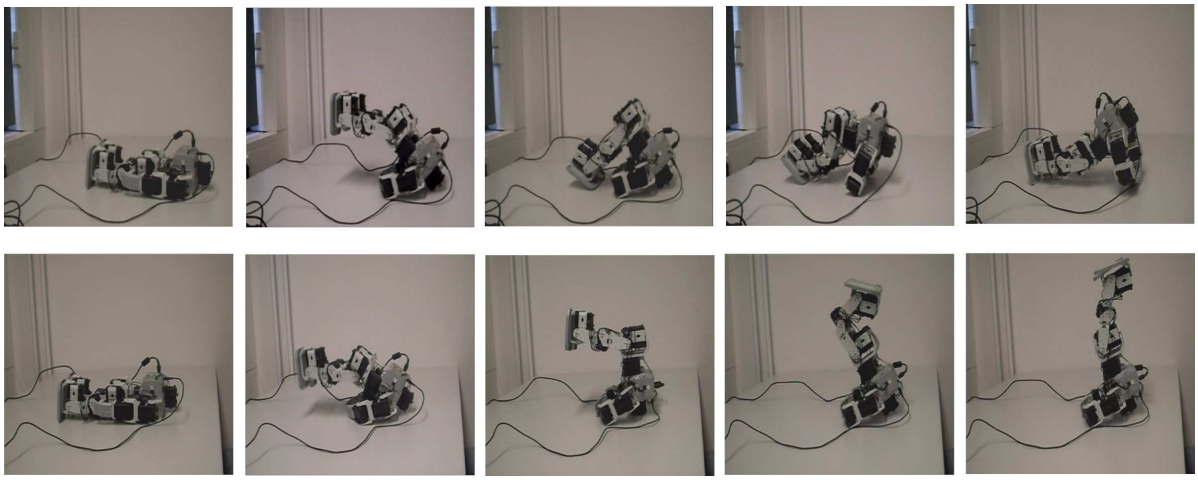


Figure 6.18: Top: A recorded headstand movement is replayed without any optimization. The robot fails to successfully perform the motion. Bottom: After optimization the robot successfully realizes a headstand.

after optimization the robot succeeds in performing a headstand even though the fitness function does not give any precise information about *how* the motion must be modified.

6.4. Programming Robots by Physical Interaction

Robot technology has progressed greatly from large, unsafe manufacturing machines to highly sophisticated androids with human-like appearances. As this technology continues to improve, the application domains of robots continue to expand, moving deeper into the realm of everyday life. Thus far, the most common type of robot, i.e. the industrial robot, has primarily inhabited dedicated working environments in factories. For a human, entering such a workspace can result in severe injuries. Recent robotic developments, however, are increasingly targeted at domestic environments and assistive tasks, in which human-robot interaction is indispensable. Several requirements must be met for humans and robots to share a common living environment. First, all physical contact between the interacting partners must be safe; the human must never be harmed. Next, the robot must be able to adapt its motions to the environment and to the actions of the human partner. Ideally, the robot should also learn from previous interaction experiences and modify its behavior according to feedback provided by the human partner.

In the following, we will investigate a physical human robot interaction scenario with a tight coupling between the human instructor and the learning robot. A test subject is asked to physically assist a state-of-the-art robot in a standing up task. Both the human and the robot are required to adapt their behaviors in order to cooperatively complete the task.

The primary technical difficulties of this approach are as follows:

- Guaranteeing safety at all times.
- Ensuring that the robot reacts appropriately to the force applied by the human instructor.
- Improving the behavior of the robot using a machine learning algorithm in a physical human robot interaction.

In order to ensure safety, the robot is equipped with pneumatically actuated flexible-joints. This allows the robot joints to have high flexibility in response to externally applied forces. Furthermore, the robot can safely apply a force to the human instructor in order to provide efficient physical assistance. This interchange of force is considered as one of the most important aspects in close physical human robot interaction.

In order to improve the behavior of the robot, we will employ imitation learning as already used in earlier scenarios. In contrast to our previous examples, however, the fitness function will be replaced by a human evaluation. After each trial, the human interaction partner judges whether the interaction was successful or failed. This judgment is used to perform optimization and update the behavior of the robot. As learning progresses, the robot's PLDPM implicitly includes the actions of the human counterpart. In this framework, refining the motion of the robot in physical interaction requires the motion of the human to be improved because the two motions influence each other. Therefore, the human counterpart is part of the learning system and its overall dynamics. The most significant difference between physical interaction learning and the earlier introduced kinesthetic bootstrapping, is that the human counterpart is included in the system. This can generate complex physical human-robot interactions and provides high adaptability.

6.4.1. Physical Interaction Learning Approach

The goal of interaction learning is to improve the cooperation of humans and robots *while* they are working to achieve a common goal. Figure 6.19 shows an overview of the learning scheme used here. This scheme can be regarded as a variation of our three-step imitation learning approach which is executed in a loop. After an initial physical interaction between a human and a robot, the human is given the chance to evaluate the behavior of the robot. More precisely, the human can judge whether the interaction was successful or not. The feedback can be provided in various ways, such as through touch or through a simple graphical user interface. Once the evaluation information is collected by the robot system, it is stored in a database in memory. The memory collects information about recent successful interactions and manages the data for the subsequent learning step. This allows us to optimize the set of training examples used for learning in order to improve learning quality.

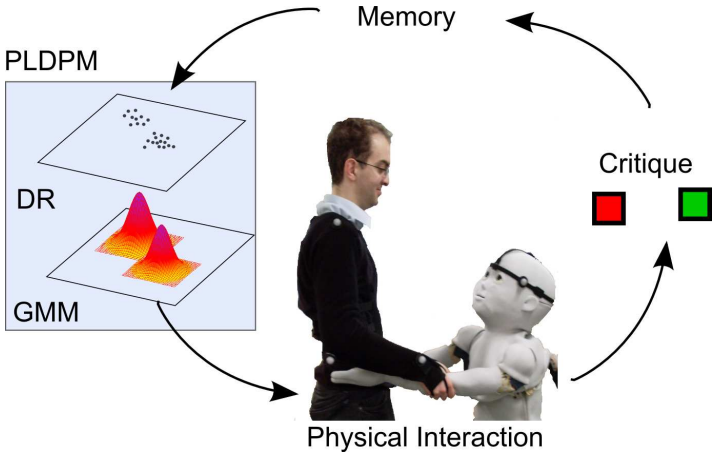


Figure 6.19: Overview of the physical interaction learning approach. After physical interaction, the human judges whether the interaction was successful. This information is stored in memory and used for later learning a PLDPM.

Figure 6.19 shows the human-in-the-loop learning system considered in this section. The behavior of the human influences the behavior of the robot, and, simultaneously, the behavior of the robot influences the behavior of the human. Furthermore, the behavior of the robot changes as learning progresses, which in turn influences the behavior of the human especially with respect to the physical support. This system demonstrates one of the applications of tightly coupled physical interaction.

After several interactions, the learning system queries the memory for a new set of training data. The data is then used to learn a corresponding PLDPM. In contrast to our earlier examples, however, a PLDPM containing three Gaussian mixture models—one for each phase of the robot's stand-up task–is learned.

Once the state vectors are projected onto a low-dimensional posture space, the resulting points into sets are grouped according to the action performed in that state. Thus, we obtain for each possible action a set of states in which the corresponding action is to be triggered. For each action, a Gaussian Mixture Model (GMM) is learned which encodes a probability density function of the learned state vectors. By computing the likelihood of a given low-dimensional state vector $\widetilde{\boldsymbol{q}}$ in a GMM of action A, we can estimate how likely it is that the robot should perform action A when in posture $\widetilde{\boldsymbol{q}}$. The learned mixture models are then used during the next physical interaction trial to determine the actions of the robot. Here, each new posture is projected into the low-dimensional posture space. Then, the likelihood of the projected point for each GMM is computed. Following a maximum-likelihood rationale, the action corresponding to the GMM with the highest likelihood is then executed by the robot.

With each iteration of the learning loop, the robot adapts its model more and more toward successful interactions. The result is a smoother and easier cooperative behavior between the human and the robot.

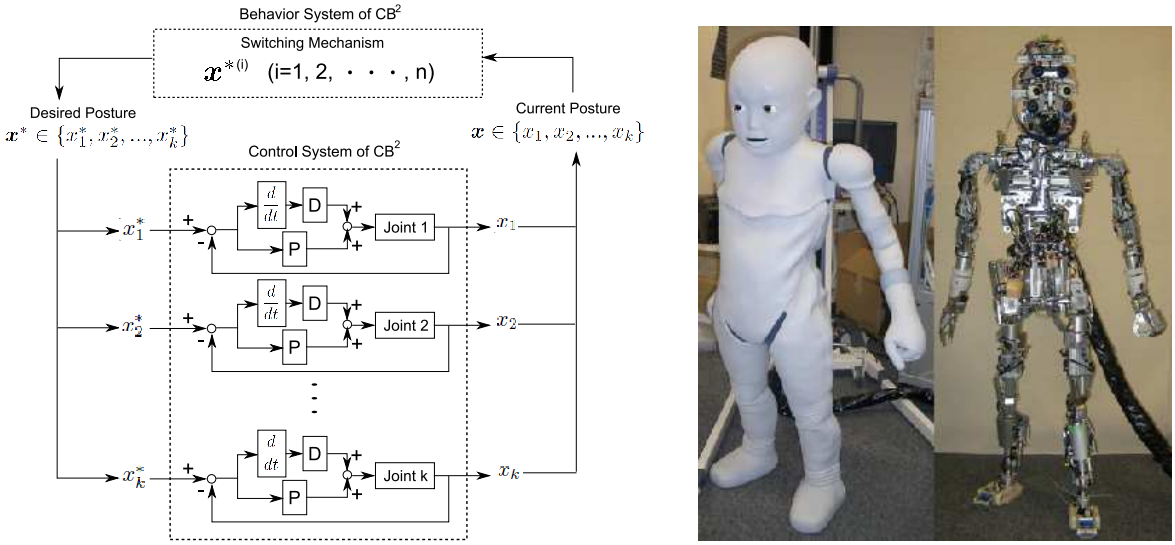


Figure 6.20: Left: Control architecture of the CB^2 robot. The desired posture is encoded as a vector $\boldsymbol{x}^*$ of angular values. Using a PD-controller, drive torques are generated in order to attain the desired posture. The switching mechanism changes between a set of different desired postures in order to achieve complex robot motions. Right: Flexible-joint humanoid robot used in the experiments in this section.

6.4.2. The CB^2 Robot

We used in the present study the so-called 'Child-robot with Biomimetic Body' or CB^2[MYN+07]. The robot features the following characteristics:

- Height: 130 cm, mass: approximately 33 kg.
- Degrees of freedom: 56.
- All joints, apart from the joints used to move the eyes and eyelids, are driven by pneumatic actuators.
- All joints, apart from the joints used to move the fingers, are equipped with potentiometers.

The joints have low mechanical impedance due to the compressibility of air. The joints can also be rendered completely passive if the system discontinues air compression during robot motion. This helps the robot to perform passive motions during physical interaction and helps to ensure the safety of the human partner. This contrasts with most other robots, in which the joints are driven by electric motors with decelerators.

The flexible actuators enable the joints to produce seemingly smooth motions, even when the input signal changes drastically. This feature of the CB^2 robot is used to realize complex motions using the simple control architecture [IMI08] depicted in Figure

6.20. More specifically, full body motions of the robot are realized by switching between a set of successive desired postures. Furthermore, the flexible actuators enable motions generated by this simple control architecture to be changed adaptively in response to an applied force from the human partner. Each posture is described by a posture vector $\boldsymbol{q}$, with each entry of the vector denoting the angular value of a particular joint. A low-level controller is implemented by PD-control of angular values. Each time the desired posture is switched drastically, large drive torques are generated, resulting in an active force being applied to the human caregiver. As the posture of the robot approaches the desired posture, the passive motion gradually becomes the dominant motion of the robot because the amount of error in the angular control gradually decreases.

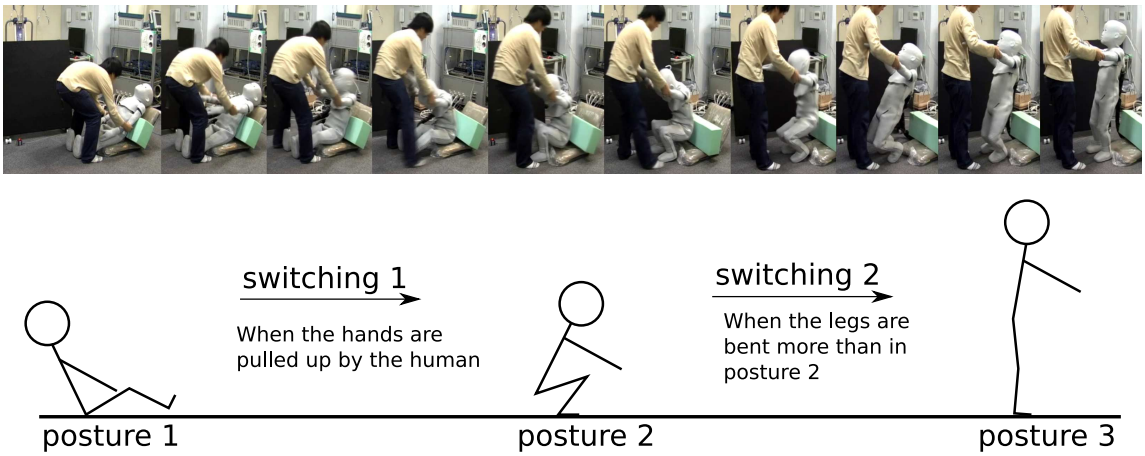


Figure 6.21: Top: A series of snaphshots showing the human-robot interaction during the standing up task. Below: The three desired postures used in the standing up task of the experiment. The learning task is to determine the ideal switching conditions between the desired postures.

Figure 6.21 shows how the standing up task investigated in our work is realized using the proposed control architecture. The behavior is realized by switching between three desired postures. At first glance, the specifications of the robot motion appear to be extremely simple. However, the switching times are highly dependent on the human interaction. More specifically, the switching times depend on the anatomy and skills of the human. This means that the robot has to adapt the switching times to the characteristics of its partner during the period of interaction.

6.4.3. Learning Method

For the standing up task the goal of learning is to determine the ideal timing for the switching actions $\boldsymbol{q}^* \in \mathcal{Q}^* \subseteq \mathcal{Q}$ between different desired postures. Here, $\boldsymbol{q}^*$ is a desired posture, $\mathcal{Q}^*$ is a set of desired postures prepared for control, and $\mathcal{Q}$ is a space that is constructed from all joint angles. The learning process constructs three different

Gaussian mixture models: one for the case in which no switching occurs, one for the first switching action, and one for the second switching action.

At each time step of the interaction between the human and the robot, the realized posture and the current desired posture of the robot are recorded. One such posture $\boldsymbol{q}$ is a 52-dimensional vector that encodes the current angular value of each joint. After the interaction is complete, the postures are stored in a database in the memory. The database holds the information for the last ten interactions. Although there are several possible ways to integrate new data into the database, the general policy used here is "new data overwrites old data, and successful interactions overwrite failed interactions".

After ten interactions, the training data is used for learning. The goal of the learning process is to construct a model that indicates when the robot should switch actions by changing the current desired posture. This rule is described by a mapping from the current posture of the robot to the desired posture that the robot should use. This is realized using GMMs. Therefore, the objective model of the learning is a probabilistic model that indicates the likelihood of desired postures in the current state.

When learning a PLDPM, we first apply dimensionality reduction to all recorded postures to construct a posture space $\mathbb{P}$. Next, we compute a GMM for each of the three switching classes. For this, we divide the projected data points into distinct sets. If no switching occurred, then the corresponding point is assigned to the first data set. Otherwise, the point is assigned to one of the other two sets corresponding to the switching phase. For each set of projected points, we learn a probability density function by a weighted sum of K Gaussian distributions. After learning, we end up with three GMMs coding three probability density functions in $\mathbb{P}$, namely, Φ_1, Φ_2, and Φ_3. Each probability density function can be used to determine the probability of a point in low-dimensional posture space with respect to a particular switching action. For example, computing $\Phi_2(\widetilde{\boldsymbol{q}})$ for a given projected robot posture $\widetilde{\boldsymbol{q}}$, returns the likelihood of the robot having to switch from the second to the third desired posture when the robot is in state $\widetilde{\boldsymbol{q}}$.

When the next experimental run is started, the robot can use the newly learned model to determine its current state and the desired posture. The current joint values are projected onto the learned low-dimensional posture space. The result is a L-dimensional point. The desired posture for the next time step $\boldsymbol{q}^*_{next}$ can be computed in a maximum-likelihood fashion as follows:

$$\boldsymbol{q}^*_{next} = \boldsymbol{q}^*(i) \in \mathcal{Q}^* \quad \text{with} \quad i = \underset{i \in \{1,..,3\}}{\operatorname{argmax}} \ \Phi_i(\widetilde{\boldsymbol{q}}) \tag{6.1}$$

In each step of the control loop, the robot calculates $\boldsymbol{q}^*_{next}$ and passes the angular values to a low-level controller. The controller then computes the joint torques needed to achieve this posture. After the interaction is complete, the human evaluation information is collected and used to update the database. The learning loop is then repeated. Figure 6.22 shows an example of such a set of interactions projected onto a low-dimensional

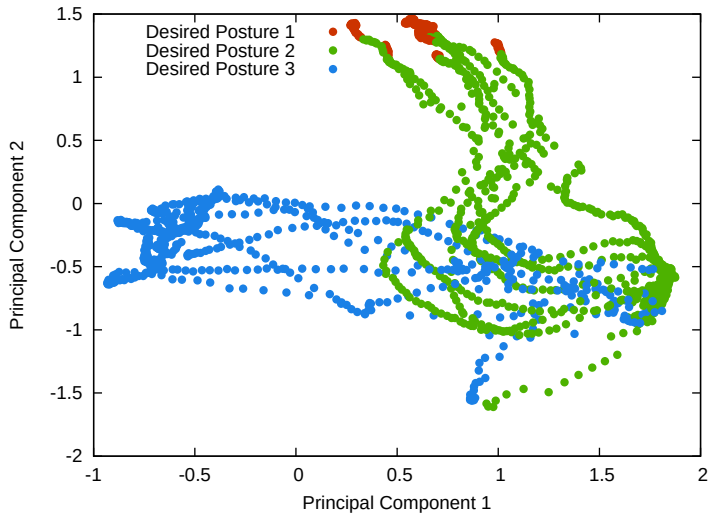


Figure 6.22: Interaction data for the standing up task projected into a low-dimensional posture space. Each point corresponds to a specific posture of the robot.

space. Each point in the plot represents one posture of the CB^2 robot during an interaction. The points are colored according to the desired posture that is active during the particular time step.

6.4.4. Experiment and Results

In order to investigate tightly coupled adaptation and the learning of a corresponding PLDPM as explained above, we performed a physical human-robot interaction (PHRI) experiment using the interaction for the standing up task introduced earlier. In particular, we considered the following question: "Does the learning algorithm lead to a symmetric learning process, in which *both* human and robot adapt their behaviors?" Furthermore, we were interested in the contribution of the learning algorithm to the improvement in the interaction. This required a careful design of the experiment, which allows us to distinguish between learning-based adaptation and adaptation due to human habituation to the robot.

The experiment was split into three independent parts. Throughout the experiment, five subjects were asked to repeatedly assist the robot in standing up. In the first part, after every ten trials, the accumulated data in the database was used for learning a new model, according to the learning scheme described in Section 6.4.3. In total, 30 interactions with two intermediate learning steps were thus performed. In the second part of the experiment, learning by the robot was disabled and fixed time steps were used for switching between the postures. In this baseline scenario, the only type of adaptation possible was the adaptation of the human to the robot. In the third and final part, learning was once again enabled. This experimental design ensures that we

Figure 6.23: Sequential photographs of the first (top) and last (bottom) interactions of the test subjects with the robot. The white curve depicts the change in position of the robot's hips. The center photograph of each sequence shows how the robot learns to maintain firm contact between its feet and the ground for both subjects.

generate baseline data that allows us to compare the results of the interactions with and without learning. In addition, by performing the baseline experiment *between* the learning experiments, we ensure that the user is already familiar with the robot. Thus, we rule out any distortion of the baseline result due to unfamiliarity.

In a preliminary experiment, we empirically confirmed that the 52-dimensional posture vector of the robot can be expressed by two principal components in order to apply proposed learning system to the standing up task. Therefore, we use dimensional reduction to reduce the posture vector to a two-dimensional vector.

Figure 6.23 shows sequential photographs of the interactions of two test subjects. The upper row of pictures shows an initial interaction, whereas the lower row of pictures shows an interaction after learning. The white dashed line indicates the height of the hips in each snapshot. In the figures we can observe a smoother transition of the hip height after the learning interaction, as compared to that before the learning interaction. In particular, the center photographs reveal strong contact between the feet and the ground and an increased hip height after learning, in contrast to the poor contact with the ungainly leg posture before learning. Since the degree to which the human helped the robot in the task and the evaluation of the robot performance are somewhat subjective, in our evaluation we focus only on whether the robot motion is refined to the degree that inefficient and jerky motions are avoided.

Figure 6.24 shows the interaction trajectories for a user before and after learning. Each trajectory was computed by projecting the robot postures into the low-dimensional posture space. Before learning, the trajectories contain loops and are partially linear. These linear pieces of the trajectories are due to jerky movements and large changes in the robot postures. In particular, for the first user, the variance in the trajectory decreases after learning. The trajectories become more similar and take on a V-shaped

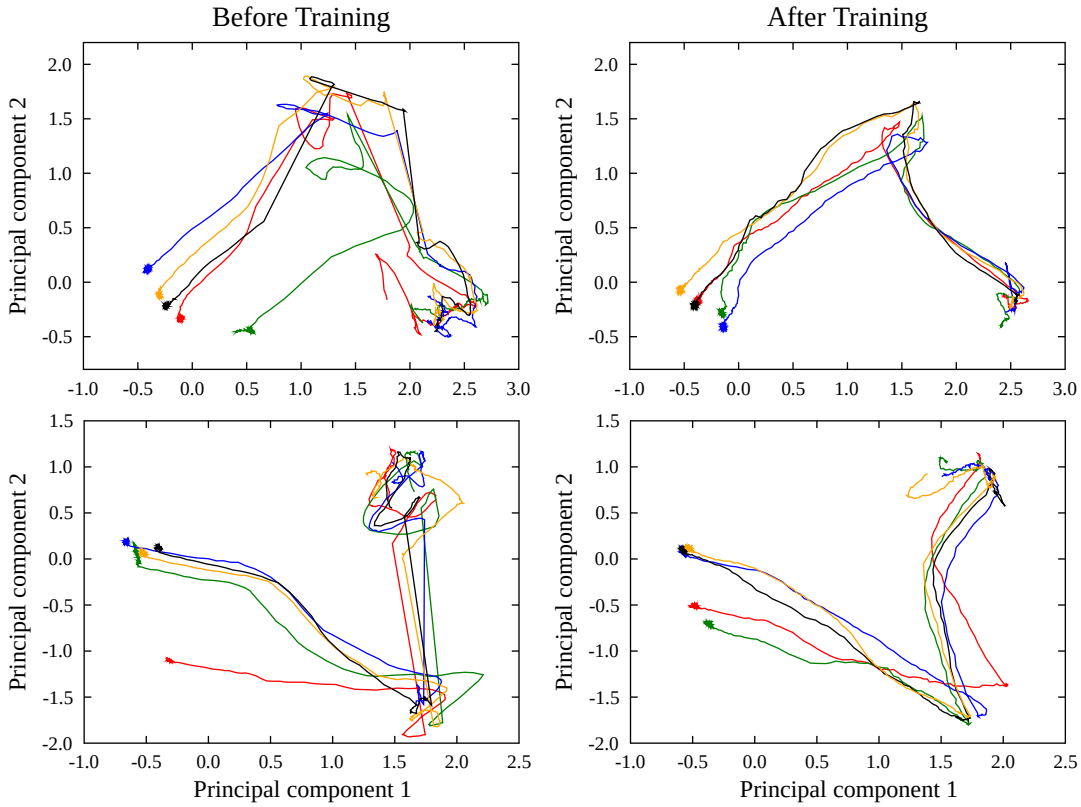


Figure 6.24: Projected interactions in the low-dimensional posture space. The upper two plots show the interaction trajectories for the first subject before and after learning. The bottom plots show the interaction trajectories for the second subject. In both cases the trajectories become smoother after learning, and sudden jumps and knots are reduced. Furthermore, the trajectories become V-shaped, clearly indicating smooth transition between the three desired postures.

form. This can be explained by the fact that the interaction consists of three desired postures. Therefore, in successful trials, the interaction leads the robot from a starting posture to an intermediate posture and then to a final posture, as shown in Figure 6.21.

In a low-dimensional space, the result is a V-shaped or triangular-shaped trajectory. This allows us to qualitatively evaluate the efficiency and naturalness of the interaction by analyzing the smoothness and shape of the low-dimensional trajectories. For example, in the case of the second subject, the trajectories before learning contain large loops at the point $(1.7, -1.5)^T$, which is the low-dimensional coordinate of the second desired posture. This phenomenon can easily be explained by taking the previous analysis into account. In the initial trials, the robot has poor contact with the floor and the legs are often not symmetrically arranged when reaching the second desired posture. As a result, lifting the robot becomes more difficult for the human, and involves slight modifications

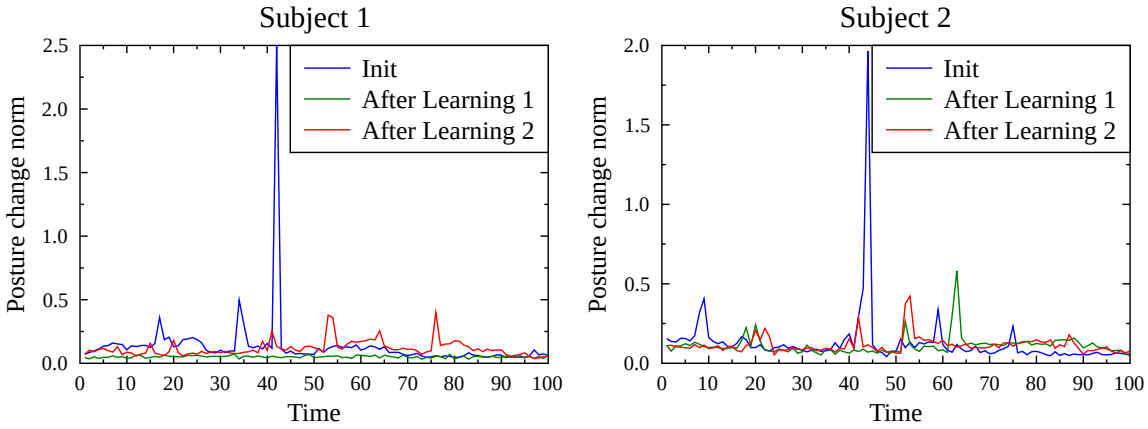


Figure 6.25: Evolution of the posture change norm during one learning experiment. The solid, dotted, and dashed lines show the evolution of the value before the robot started learning, after the first intermediate learning step, and after the second intermediate learning step, respectively.

of the robot posture in order to make the feet more stable. This interrupts the flow of the standing up task and increases the interaction burden for the human caregiver.

In order to confirm the above findings we quantified the robot motion using the *posture change norm*. The posture change norm a of the robot motion was calculated using the Euclidean distance between the data points of t and $t-1$ in the set $\mathcal{Q}$ defined using each joint angle as a base:

$$a_{(t)} = \| \boldsymbol{q}(t) - \boldsymbol{q}(t-1) \|_2, \qquad \boldsymbol{q}(t), \boldsymbol{q}(t-1) \in \mathcal{Q}. \tag{6.2}$$

Computing the posture change norm at each time step of the interaction results in the time series depicted in Figure 6.25. The solid line shows the posture change norm during the initial interaction phase. We can see a sudden peak indicating a large change in the robot posture and, consequently, a non-smooth motion. This is not desirable, because large changes in the robot posture result from strong forces acting on the robot. The other lines show the evolution of the norm after each learning step. With each learning step, the number of peaks in the time series is reduced. In other words, the fluctuations in the posture change norm decrease leading to a smoother and more efficient motion.

A statistical analysis of the data further underlines our hypothesis. We computed the mean value and the standard deviation of the sum of the posture change norm during the interactions. Figure 6.26 shows the evolution of these values with each learning step. For all subjects, we see that the mean and standard deviation of the posture change norm decreased as the experiment progresses. In the baseline experiment, only one subject was able to significantly improve the interactions, where statistical significance is computed using a t-test. None of the other subjects were able to improve their interactions. In the first experiment using the proposed learning system is used, three

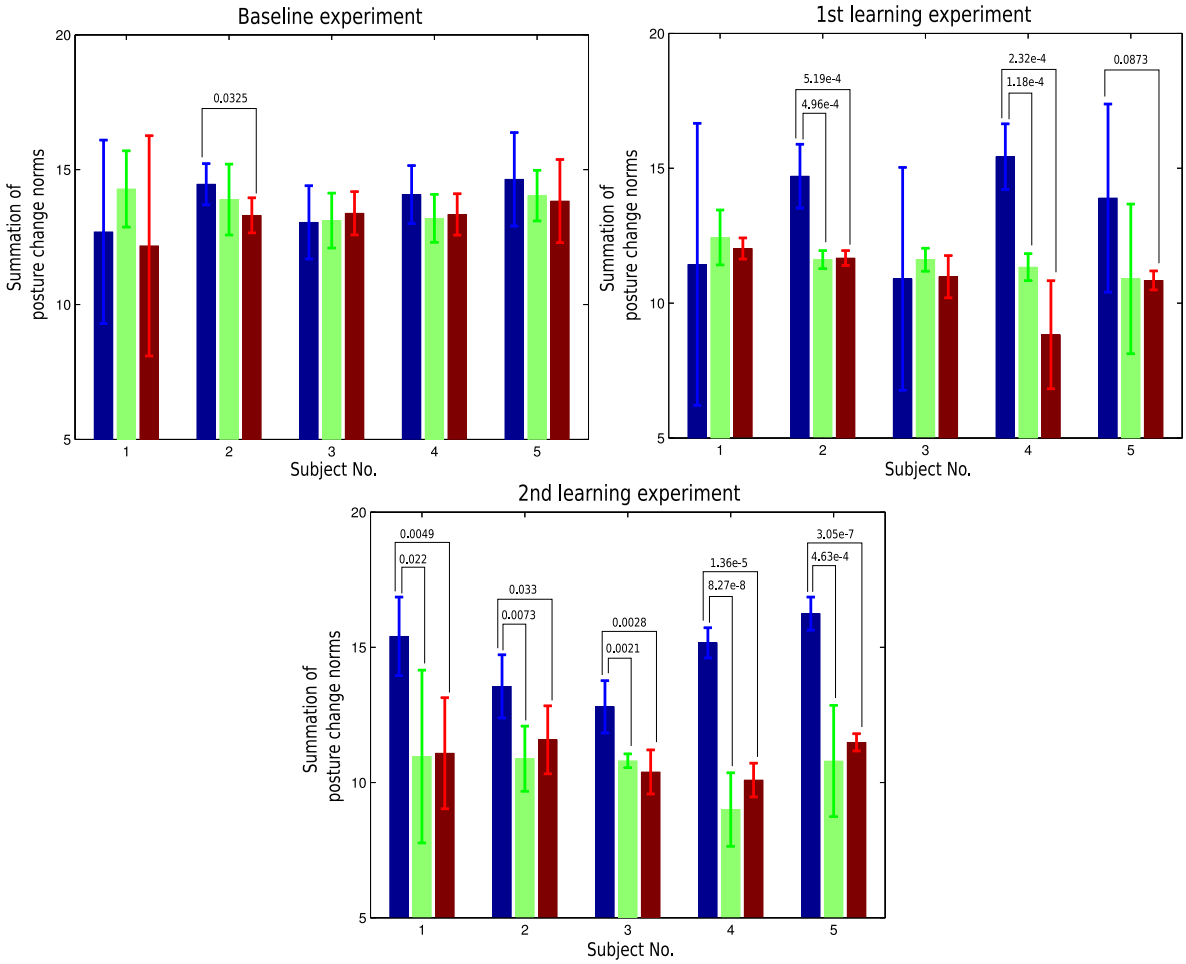


Figure 6.26: Mean and standard deviation of the summation of the posture change norm of test subjects in the baseline experiment (left), the first learning experiment (middle), and the final training experiment (right). The blue, green, and red bars indicate the mean and standard deviation values during each of the intermediate learning steps (after every 10 trials). In the case of the baseline experiment, only subject 2 shows a significant improvement after all trials. In the first learning experiment, subjects 2, 4, and 5 show significant improvements. In the final experiment, the interaction with the robot improved for all subjects. With each learning trial, the indicated values decrease, and the movement of the robot becomes smoother and more synchronized with that of the subject.

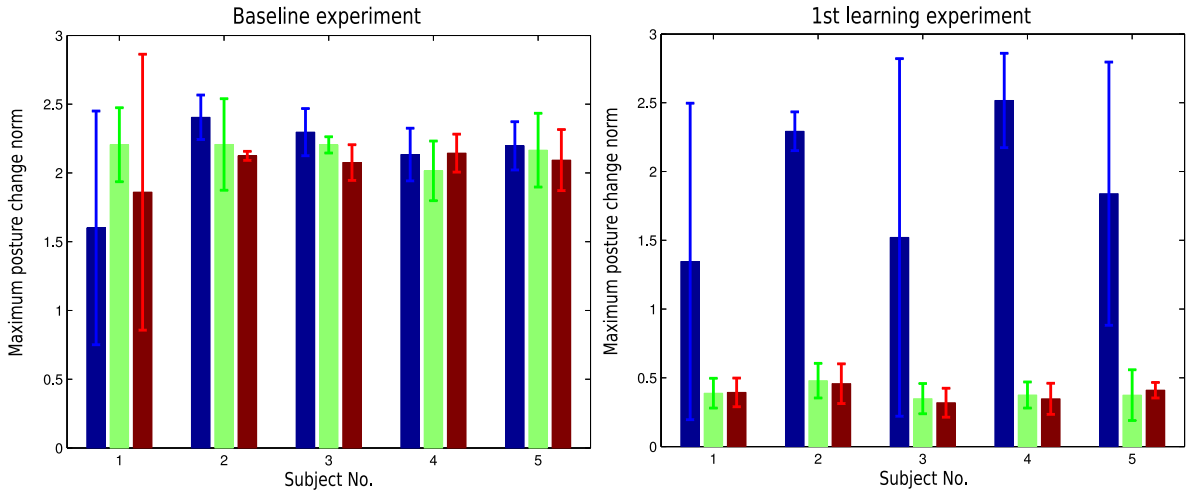


Figure 6.27: Change in the maximum posture change norm in each phase during the baseline experiment and the first learning experiment. No significant difference in the maximum posture change norm is observed in the baseline experiment. On the other hand, there are large changes in the maximum posture change norm in the learning experiment. For all subjects, the values decrease drastically after learning.

subjects show significant improvement. Finally, in the second learning experiment, all of the subjects showed significant improvement in their interactions. This indicates that while a human can adapt to a robot and thus improve their interactions (as in the baseline experiment), this adaptation can be significantly improved by enabling the robot with learning capabilities (first and second learning experiments). We also analyzed the maximum values of the posture change norm during the interaction. Figure 6.27 shows the change in the value of the maximum posture change norm during each learning phase of the baseline experiment and the first learning experiment. No significant difference in the maximum posture change norm is observed in the baseline experiment. On the other hand, in the learning experiment, large changes in the maximum posture change norm occur. For all subjects, these values drastically decrease after learning.

6.4.5. Discussion

Our experiments lead us to the following observations: First, when learning and adaptation is only possible on the side of the human caregiver, generally, little or no improvement is measured. However, even in this asymmetric learning situation, at least one subject was able to adapt to the robot so as to significantly improve the interaction quality. This shows the human ability to adapt quickly to new situations and motor tasks. Second, the interaction quality significantly improved in the first learning experiment, and the improvement was even more remarkable after the second learning experiment. These results support our working hypothesis that the proposed learning

system significantly improves PHRI. Another interesting observation is that the human adaptation to the robot occurred both in the short and in the long term. At the beginning of the experiment, the users were intimidated by the robot and the experimental setup. During the course of the experiment, however, the test subjects became more and more comfortable with the situation and the robot dynamics. As a result, the test subjects found it easier to interact with the robot. This suggests that while learning algorithms are needed for improving the PHRI, they can be rendered more efficient if familiarization of the human with the robot is also taken into account. A special familiarization phase, in which the human caregiver becomes accustomed to the robot before any cooperative tasks, might be one approach. Another method by which to familiarize the human with the robot might be a well designed interaction protocol that involves tasks that are intended only to familiarize the human with the robot. An interesting feature of the proposed procedure is the ability to monitor the progress of learning as trajectories in a low-dimensional space. The results of the present study indicate that the trajectories converge toward a V-shaped pattern for the standing up task. Furthermore, the trajectories after learning appear to show particular points or bottlenecks through which they pass. This is reminiscent of the research by Kuniyoshi and colleagues on the 'knack' of motions (for a recent publication on the topic see [OSOK09]). They showed that dynamic motions for a particular task often have a bottleneck in the state space. This bottleneck is the result of the interaction of the human body and the environment. Kuniyoshi and his colleagues referred to this property as *knack*, and showed that the knack can be exploited in order to efficiently control a humanoid robot. In the proposed PHRI scenario, the dynamics of the robot strongly depends on the dynamics of the human caregiver. A knack appears in PHRI because of the strong coupling between the human and the robot and the resulting joint dynamics. In other words, the human can be regarded as a changing environment that constrains the robot dynamics. Note that, although only the posture of the robot was used to create the trajectories, we can still discern a knack that is based on joint dynamics. However, it can be argued that posture information is not sufficient to draw final conclusions about the joint dynamics.

To address this question, we also applied our learning method to a different cooperative PHRI task, namely that of *assisted walking* as can be seen in Figure 6.28. In this scenario the human caregiver must assist the robot while the latter is trying to walk. Our early results show that slow assisted walking can successfully be realized using the proposed physical interaction learning technique. However, the robot often failed to keep up when the human demonstrator increased or reduced the speed of his walking gaits. This is due to the reactive nature of estimating the joint dynamics from the postures only. To keep up with a human interaction partner in this scenario, the robot must be more predictive in his estimation of the joint dynamics. One possible approach to overcome this problem, is to include sensor information into the probabilistic low-dimensional posture models. That is, the state of the robot would be based on the current joint angles as well as the information gathered from the sensors under the skin. In this case switching between one posture and another would also be influenced by the amount of pressure exerted by

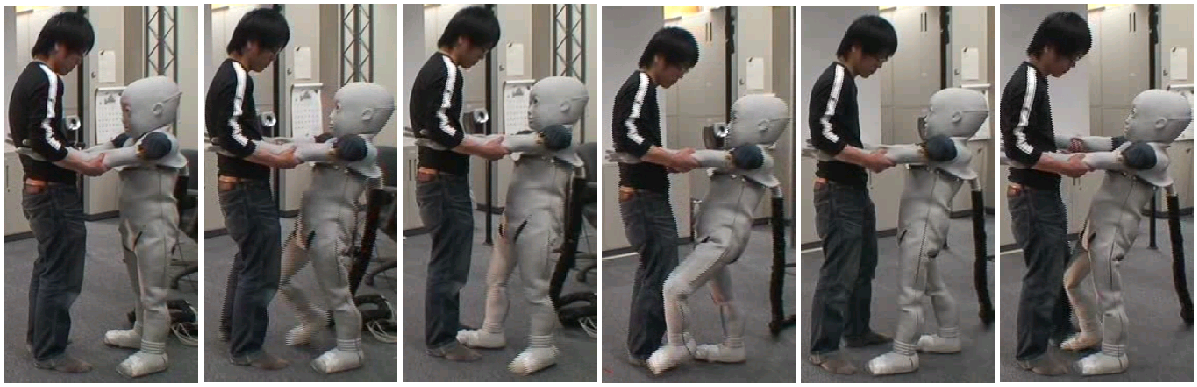


Figure 6.28: A human caregiver assists the robot in his attempt to perform several walking steps. The movements of the robot are adapted by the physical interaction learning method presented in this chapter.

the human caregiver on the robot's body, e.g. the arms during assisted walking. An interesting aspect in this regard, is the question of whether the sensor information can simply be concatenated with the joint values to form the new training vectors, or if we have to add a special treatment for this type of data. Further studies are underway in order to obtain a conclusive answer to these questions.

6.5. Other Approaches

In computer animation spacetime optimization [WK88] has been proposed for the adaptation of full-body motions to new environmental conditions (see Chapter 2). Spacetime optimization can effectively be used for synthesizing walking animations from footprint constraints. However, in contrast to the approach presented in this chapter, spacetime optimization does not include methods for incorporating naturalness of postures and movements into the synthesis process. As a result, it can not be guaranteed that a generated animation is realistic and visually appealing. Additionally, this approach is limited to simple kinematic constraints and cannot be used for more complex adaptation processes. A different technique for adapting a recorded motion capture animation was presented in [ZH02]. The proposed technique uses a combination of motion capture driven animations with dynamics simulations and inverse kinematics. Motion capture ensures that the generated animations retain their subtle details of human motion. Simulations of dynamics, on the other hand, ensure that a virtual human is influenced by forces coming from the environment (passive reactions). Finally, inverse kinematics is used to ensure that a particular position can be reached by the virtual human. The approach is especially well-suited for reaching or boxing tasks, however, not suited for more complex tasks, such as the adaptation of a recorded walking motion to a new synthetic humanoid (as described in Section 6.3).

Several researchers have also applied dimensionality reduction techniques, and in particular PCA, to the problem of motion synthesis and adaptation. An early application of PCA to animations was presented in [AM00]. Here, PCA is mainly used to compress a given animation and represent it at different levels of detail. In [GBT04], a PCA-based walking engine was proposed. The engine is able to produce walking gaits at different speeds and for virtual humans with different heights. However, the engine does not include methods for generating walking gaits from footprint constraints. A similar approach was used in [CGGR07] to synthesize walking gaits for a small humanoid robot. Both works target walking skills only and do not address the general question of how to imitate and adapt arbitrary motor skills. Another application of dimensionality reduction techniques for the imitation of motions was presented in [JM03]. A recorded movement is automatically segmented into distinct behaviors. These behaviors constitute a 'vocabulary' from which new animations are later generated by concatenation. In contrast to our work, however, the approach by Jenkins and colleagues is limited to gestures and non-goal directed motions. This restriction significantly constrains the application domains of the approach.

6.6. Conclusion

In this chapter we to investigated whether our imitation learning approach scales up to full-body motions and whether it can meet the requirements of different domains, e.g. computer animation or robotics. To this end, we first investigated how locomotion skills can be taught to virtual humans. Using motion capture data, a PLDPM was learned for walking and stair climbing. The results showed that our approach is scalable for full-body motions. It successfully synthesizes animations that meet the constraints posed by the anatomy of the virtual human and the demands of the environment. In the second part of the chapter, we focused on robotic agents, physical environments and kinesthetic training data. Instead of motion capture, we let the user move the joints of a robot and collect the resulting sensory data. Even with such a considerable change of setup, we were still able to use exactly the same three-step approach as in our earlier experiments. Finally, in the last part of this chapter, we presented a physical human-robot interaction scenario in which successful task completion can only be achieved through coordinated actions between the robot and a human involving physical contact. In contrast to previous research in this field, the robot used in this thesis is in close physical contact with the human partner and plays an active role during the cooperative task. The CB^2 robot, through its flexible-joint design and soft silicone skin, is particularly well suited for such tasks. Because of these features, physical interactions become more 'natural' and lifelike. In an experiment inspired by the parenting behavior in humans, we were able to show that the proposed learning method results in measurable improvements of the interaction. At the same time, we note that the human's familiarity with the robot has an impact on the interaction.

An interesting conclusion of this chapter, is that fitness functions can be specified at different levels of granularity or abstraction. They can even be replaced by human judgement, as in the case of the physical interaction learning. When synthesizing locomotion skills for virtual humans, we included each step into the calculation of the fitness. At one hand this guarantees high control over the resulting animation, on the other hand, however, it increases the specification burden for the user. The location of each step must be determined prior to the start of the optimization process. In contrast to that, when optimizing the walking skill for the small humanoid robot, we only used the travelled distance as a fitness function. That is, we only cared about the result of the behavior not the means by which to achieve it. In traditional evolutionary optimization the usage of such abstract fitness function can have severe implications on the synthesized results. Instead of human-like walking, the evolutionary process might result in a limping, hopping or even quadruped crawling gait. Our approach overcomes this problem by using a behavior-specific posture space as learned by the PLDPM, e.g. the posture space does not contain poses for quadruped walking. Additionally, the evolutionary algorithm is not started from a random population as traditionally done. Instead, the demonstrated example motion is taken as a template, from which the individuals of the first population are created through small perturbations, i.e. mutation. The evolution of walking skills has attracted considerable attention in robotics, and in particular in the RoboCup community [KAK+97]. In fact, some teams in the RoboCup competition, for instance the team of the *Humboldt University* [HHB07], use the same robot platform as used in Section 6.3 of this thesis. Our results are therefore directly applicable to this domain.

Our imitation learning approach works on the level of postures and joint angles and does not explicitly specify how these joint angles are realized. In absence of physics (such as the case of our virtual human example) the solution to this problem is trivial: we can overwrite the current joint angles with any desired values. However, as we have seen in the case of our simulated and real robots, this can not be done in environments with physical properties. In these environments we used simple PID controllers to compute forces for realizing the desired joint configurations. The use of PID controllers was mainly employed for explanatory purposes. More accurate and sophisticated techniques can be used for this purpose. For example, in our work in [BIT+07] we have proposed a neural network based learning framework that is specifically designed to realize high accuracy, low-level robot control. The framework was specifically tailored to the demands of modern android robots.

One difficulty that we encountered in this chapter is often referred to as the *'reality-gap'* in evolutionary robotics. As stated in [ZRV04]: "No matter how accurate a simulator is, there is always substantial difference between simulation and reality, the so-called reality-gap". Hence, there is no guarantee that a behavior optimized in simulation will also work in reality. Our solution to this problem is to try out the fittest individual from different generations. Thereafter, we use the controller which produce the most stable motion of the real robot. While this does not solve the problem completely, it proved to be a practical, and in most cases successfull approach in our experiments. Generally,

there is no final solution to the reality-gap problem. However, in the literature there are some interesting approaches that try to overcome the gap. For example in [LBZM06], both the robot and the simulator were evolved. For the evolving simulator, the fitness function evaluates the difference between simulated and recorded observations of the robot.

In summary, this chapter showed that the proposed imitation learning approach scales up to the imitation of full-body motions. Using this approach, we can generate highly dynamic motions such as walking, standing-up, or even the performance of a headstand. What is missing so far, however, is the ability to combine several behaviors in order to produce long sequences of actions. In the next chapter, we will address this issue and present an example application domain in which the imitation of complex action sequences is beneficial.

7. Learning to Imitate Complex Action Sequences

So far, the techniques presented in this thesis have only addressed the imitation of *single* motor tasks. To extend our approach to the imitation at the level of action sequences, we will subsequently introduce the *action capture* method [JAHW06; JAHV10]. Action capture was developed within the *'Virtual Workers'* research project funded by the German research foundation (DFG). The method tracks and recognizes the interactions of a human user in a virtual environment and stores the result in an abstract, multi-layer representation. Later, this representation can be used to reproduce the user's actions by virtual characters.

Different aspects of the action capture method have been investigated within this project. For instance, pattern recognition algorithms have been used to segment and understand the user's actions. Furthermore, the project investigated representation formalisms which allow for a compact documentation of the actions observed. These aspects have been extensively treated in [HBAJ08] and [VAHJ09]. This chapter focuses on the animation synthesis problem within action capture. More specifically, we will contribute a behavior-based animation framework which allows us to synthesize animations for complex action sequences. At the core of this framework is a set of behaviors which are learned using the PLDPM construction algorithm and the imitation learning approach described in Chapter 4.

While an exhaustive description of the action capture method is beyond the scope of this thesis, we will in the following provide a brief overview of the method and its components. Then, we will present a set of behaviors which can be combined to generate animations for long action sequences. The behaviors cover a wide range of manipulation skills. Finally, we show several example scenarios in which these behaviors are used to imitate long sequences of actions.

7.1. Action Capture

7.1.1. Motivation

Action capture is a VR-based method for the imitation of recorded action sequences by virtual characters. It was proposed and further enhanced by B. Jung and colleagues [JAHW06; JAHV10]. Similar to motion capture, the user's movements are recorded by

means of position trackers and data-gloves. In contrast to motion capture, however, not only the user's movements are tracked but also his or her interactions with scene objects. The name is motivated by the work of neuroscientist M. Arbib who stated that actions are always associated with a goal, a hypothesis he formulated as: action = movement + goal [Arb02].

A motivating application domain for action capture is known as virtual prototyping. Virtual prototyping is a modern approach to engineering, wherein a technical product is first simulated, optimized and evaluated within virtual reality, before a physical prototype is built. By working within a virtual environment, many variations and designs of the prototype can be efficiently evaluated. The analysis of the interaction between a user and a technical product is therefore an important goal. For example, in automotive engineering, the interior of a virtual car prototype is typically evaluated with respect to ergonomics.

One approach for performing such an analysis is the application of immersive virtual reality, in which a VR user performs various operation procedures on the prototype. To test the prototype's design, the user may interactively simulate the handling of the steering wheel, gear shift, radio controls, and other instruments. The advantage of this approach, is that the analysis can already be carried out in an early phase of the product's design. In addition, interaction with a virtual prototype is highly natural and realistic, and involves the same movements as for a physical prototype. Hence, important conclusions can be drawn on the basis of these experiments without the need to build a real prototype.

However, the approach bears two main disadvantages. First, the user must repeat the operation procedure whenever the design of the car's interior changes. Especially in the early design phase, when the design of the car is repeatedly modified, this can be a time consuming process. A second disadvantage is rooted in the evaluation setup which relies on the subjective experience of a single or a few VR users only. As a result of the limited group of test users, crucial insights may be missed in the analysis of a prototype.

One approach to overcome these problems is to use virtual humans for ergonomic analyses. As already seen in the earlier sections, virtual humans can come in many sizes and body proportions and allow for an arbitrarily large number of test persons. Repeating the operation procedures with a large set of virtual humans allows the engineer to draw more informed conclusions on the ergonomics of the current car prototype. This reduces the risk of omitting crucial aspects. However, difficulties arise in the animation process of the virtual humans. When animating complex, articulated 3D models via desktop GUIs, important details of human movements are easily missed and as consequence the resulting ergonomic analyses may be less meaningful.

The idea of action capture is to combine the advantages of the two approaches: First, a VR user simulates the operation of a virtual prototype using immersive VR technology, such as 6 DOF tracking devices and data gloves, then, imitation learning techniques are applied on the interaction protocols of the VR user's performance. This produces a compact description of the performed action sequences. In turn, the extracted actions

can be replayed using a variety of virtual humans. Imitation learning ensures that synthesized animations adapt to changes in the design of the prototype or the anatomy of the virtual human. Therefore, once an operation procedure is recorded and the corresponding action representation is learned, there is no need for the VR user to repeat the procedure anymore. Action capture can be seen as an extension of conventional motion capture wherein both the actors movements as well as his interactions with the objects of a virtual environment are stored.

7.1.2. Overview of the Method

In a typical action capture session the user sees a virtual prototype through a VR display, such as a Power-Wall or a CAVE. The user can manipulate the virtual prototype in a natural way by touching, grasping or changing its position. The user's movements are recorded through a set of VR input devices, including tracking devices for hand and arm movements. The configuration of a user's hand is mapped onto a virtual representation of the hand using data gloves or optical fingertracking systems. The data recorded by the tracking hardware is collected in a continuous fashion. All input devices together produce an extensive stream of heterogeneous data. This raw data is persistently stored in an interaction database for later analysis and processing. It includes among other things the trajectory of the human's arm, hand posture data and timing information. The database collects all data recorded in an action capture session.

Figure 7.1: Left: A VR user interacts with the virtual prototype of a car. Right: The user's actions are repeated by a virtual human.

This database is a central datastore and allows for later analysis of specific recording sessions as well as analyses on a larger scale, e.g. all recording sessions during the last week. However, in order to produce more meaningful descriptions of an observed action sequence, the recorded raw data must be transformed into a more abstract representation. Therefore, after acquisition of low-level tracking data, user movements and interactions

are abstracted to higher-level action representations. To this end, segmentation and pattern recognition algorithms are employed, resulting in a compact description of the primitive actions and scene objects involved in the demonstrated task. In the playback phase, these action sequences are reproduced by virtual characters using a repertoire of behaviors. These behaviors can be learned from the recorded motion data using the PLDPM construction algorithm.

Figure 7.1 illustrates the process of action capture in an example setting in which the ergonomics of a virtual car prototype is evaluated by a VR user. The interactions of the user with scene objects such as handles, buttons or the steering wheel are tracked, recognized and later stored within a multi-layered representation. The action representation can then be used to synthesize and visualize the demonstrated interactions using virtual humans. In contrast to earlier uses of virtual humans in virtual prototyping, recorded actions can still be synthesized if the design of the car or the anatomy of the virtual human is changed, i.e. there is no need for manual modification of the virtual human's movements. This gives the VR user a wide range of possibilities for performing large numbers of ergonomic analyses in a short period of time. For example, he can replay the performed actions using different virtual humans in order to draw important conclusions about the positioning of handles. Figure 7.2, for instance, shows a set of virtual humans that reproduce the aforementioned actions in a car prototype. Because of the different sizes of their limbs, the virtual humans need to take on different postures to manipulate the gearshift. Using action capture, a car designer can therefore try different positions of the gearshift before making an informed decision.

Figure 7.2: Virtual humans of different gender and with different anatomical properties replay an action previously recorded from a VR user. The animation is synthesized using adaptive behaviors.

According to [JAHW06], the *setting* for action capture consists of the following three components:

- Virtual environment: which supports its interactive manipulation by a human user. I.e. the virtual environment contains interactive objects which e.g. can be picked up and displaced, buttons that can be pushed, knobs for turning etc. Whenever the user manipulates a scene object, the virtual environment can detect this manipulation and generate a corresponding event.

- Human demonstrator (teacher): who performs an action or a sequence of actions in the virtual environment. The human teacher's actions are tracked using typical VR input devices such as position trackers and data gloves.

- Synthetic humanoid (learner): who observes the teacher's actions and learns to repeat them. The synthetic humanoid is equipped with a repertoire of *basic behaviors*, e.g. for pushing a button or grasping an object. These basic behaviors may be further parametrized, e.g. with a target position, a target object, or a hand shape to be assumed during a manipulation action.

Given these requirements, the process of action capture involves three steps (see Figure 7.3). In the first step, the teacher's movements and interactions with scene objects are tracked, segmented, and classified as action primitives (basic interactions). The goal of this process is to extract the structure of the observed action sequence and recognize the performed actions as well as their specific parameters. This involves, among other things, identifying of the name of the manipulated object, recognizing the grasp type with which the object was grasped, analyzing the spatial relationship between the user's hand and the object, as well as recording the exact timing of the manipulation process.

The outcome of the observation step is an intermediate representation of the user's actions called *interaction events*. Interaction events hold a wide range of extracted data ranging from raw tracking data (like motion capture) to more abstract information such as the used grasp type. The extraction of interaction events is an important step in the action capture process. Segmentation transforms the continuous stream of data into an ordered, discrete *sequence* of events. By analyzing the events one-by-one, we can recreate the steps needed to reproduce the user's manipulations while taking into account not only the movements but also their effect on the environment. However, especially for long interaction event sequences this type of representation becomes lengthy and, therefore, too difficult to analyze and understand. Brevity and abstraction are vital for easy analysis, modification and authoring of action sequences. Therefore, in the second step, observed action primitives are combined and stored using a high-level action representation.

The action representation of an observed manipulation process is derived by processing the generated interaction events. Sequences of interaction events are mapped onto corresponding actions using pattern recognition and pattern matching algorithms. For

example, if the user reaches out for an object and turns it, a `Turn` action will be issued. In addition to the type of action, parameters such as the object's name will be stored. Actions such as e.g. `Turn`, `Pick` or `Push` are represented in a human-readable, textual representation (see [VAHJ09] for a detailed account of the action representation language). Therefore, they can be manually postprocessed by a human editor. Even though the main goal of this approach is to store recorded manipulations during action capture, this representation can be used to rapidly prototype animations by manually creating action description files.

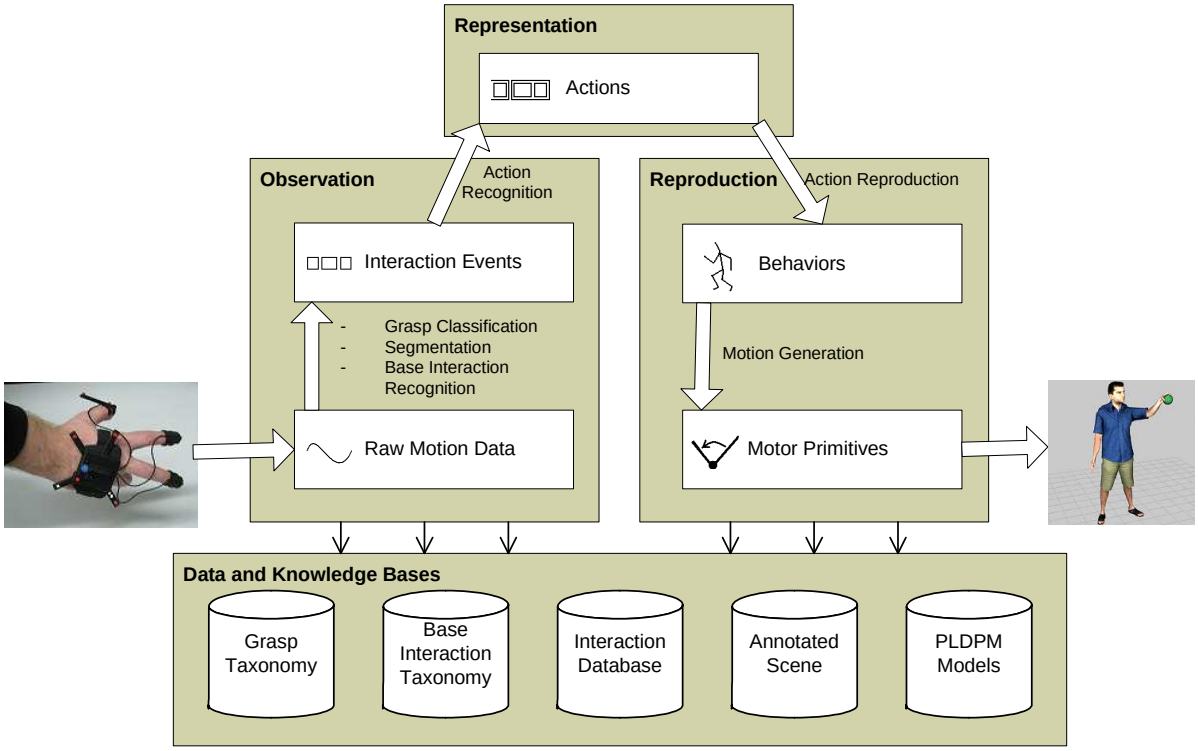


Figure 7.3: Components and system architecture for action capture.

In the reproduction step, the actions are mapped to goal-directed behaviors of the virtual character. Behaviors are responsible for motion generation and the calculation of contact conditions between the virtual character's hand and scene objects. Finally, the execution of the motor commands generated by the behaviors is then realized by motor primitives which interface to the specific character animation system in use.

To reproduce the actions performed by the user, the virtual human needs to have a sufficiently large repertoire of behaviors. We have shown in earlier chapters, how behaviors for pushing, grasping and walking can easily be learned using imitation learning and the PLDPM construction algorithm. Indeed, the information needed to learn a PLDPM is already stored in the database. For example, a PLDPM for a specific grasp type can easily be learned by querying the information about the recorded hand shapes (angular

values) and the grasp types to which each hand shape belongs. Using this information, the PLDPM construction algorithm can be used to learn models for different grasp types or for different users. The modularity of this approach allows for an incremental expansion of the virtual human's motor capabilities.

7.1.3. Example: Brewing an Espresso

To better understand the animation synthesis process of action capture, let us consider the following (hand-crafted) illustrative example: brewing a cup of espresso using the virtual prototype of an espresso machine.

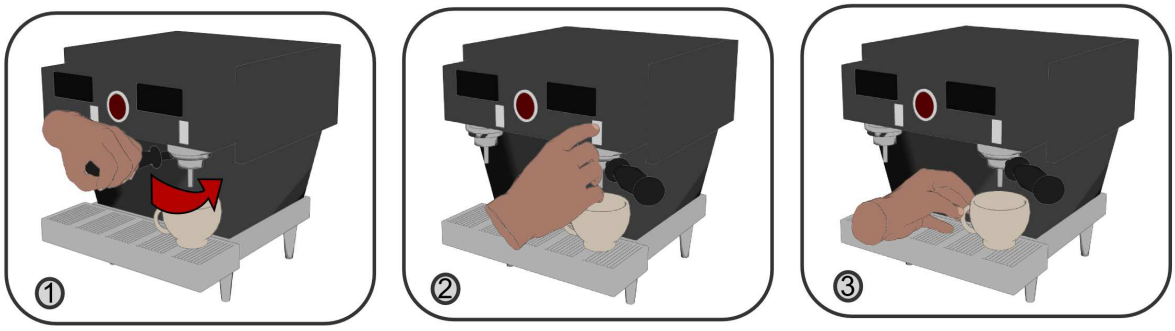


Figure 7.4: A sequence of three actions for brewing an espresso: ① Attachment of a porta filer to the espresso machine and turning it into place. ② Pressing a button for pouring in the coffee. ③ Taking out the cup from the machine. Notice, that both hands are used in the process.

Figure 7.4 schematically illustrates the sequence of actions that need to be performed by a virtual human in order to solve the task. First, a portafilter must be attached to the machine by turning the handle. Then, a button must be pressed to fill the cup with coffee. Finally, the cup must be grasped and picked up. Note, that procedure involves the usage of both hands. The portafilter is attached with the left hand, while the button is pressed with the right hand.

To generate motions realizing the above procedure, the animation synthesis process takes a description of the involved behaviors as input. More precisely, it takes an XML-representation of the behaviors, simply called *movement plan*. Such a plan can be either the result of an action capture session, or can be hand-crafted by the user. The synthesis process takes such a plan as input and generates an animation which reproduces the documented manipulations. To this end, it computes the actual joint angles and animation parameters taking into account the current state of the environment and the virtual human.

```
<?xml version="1.0" ?>
<movement-plans>
  <default-parameters>
    <param name="version">3</param>
    <param name="boneNamingType">hanim</param>
  </default-parameters>
  <plan>
    <name>Brew a cup of espresso</name>
    <behavior>
      <type>Turn</type>
      <param name="start-time">0</param>
      <param name="end-time">2</param>
      <param name="side">left</param>
      <param name="object">handle</param>
      <param name="turn-origin">-0.1 0.0 0</param>
      <param name="turn-axis">0 0 1</param>
      <param name="turn-angle">20</param>
      <param name="follow-ik-method">IkArmHeuristic</param>
    </behavior>
    <behavior>
      <type>Push</type>
      <param name="start-time">3</param>
      <param name="end-time">5</param>
      <param name="side">right</param>
      <param name="object">button_right</param>
      <param name="grasp-type">extension</param>
      <param name="grasp-end-shape">-1 0 0</param>
      <param name="follow-ik-method">IkArmHeuristic</param>
    </behavior>
    <behavior>
      <type>Pick</type>
      <param name="start-time">5</param>
      <param name="end-time">7</param>
      <param name="side">left</param>
      <param name="object">espresso_cup</param>
      <param name="grasp-type">Schlesinger::tip</param>
      <param name="grasp-object-interaction">attach</param>
      <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
      <param name="follow-ik-method">IkArmHeuristic</param>
    </behavior>
  </plan>
</movement-plans>
```

Listing 7.1: A movement plan consisting of the behaviors needed for brewing a cup of espresso.

Listing 7.1 shows a simple movement plan for the espresso machine example. The plan consists only of three behaviors: one for turning the handle, one for pushing the button and one for picking the cup. Each behavior is parametrized by its start time, end time, the hand side, the manipulated object, the grasp type, as well as additional parameters. These parameters can also be changed by hand. For example, the user might change the plan so that a different button is pressed.

Given such a plan, the animation synthesis process starts and ends the specified behaviors according to the indicated timing. Once a behavior is activated, it calculates the joint angles needed to realize the corresponding animation. This is done based on the current state of the environment. Hence, even if the target object is displaced as a result of a behavior activated earlier, the currently active behavior can take this into account and adapt the virtual human's motions accordingly. The adaptive nature of these behaviors ensures that even in dynamic scenarios faithful animations are synthesized.

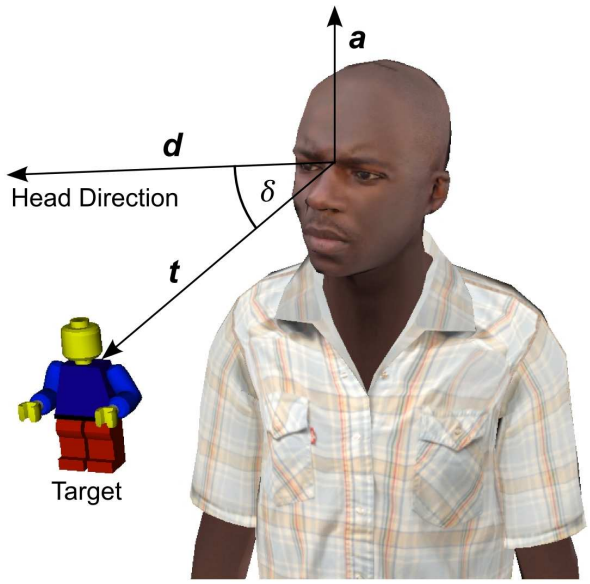


Figure 7.5: The calculation of angle δ and axis $\boldsymbol{a}$ needed to turn the head to face the target object.

7.2. Behavior Repertoire

In the following we will describe a set of behaviors which are currently implemented in the Virtual Human Framework. At the core of most of these behaviors lies a PLDPM learned using the techniques presented in Chapter 4. Note that the selection is non-exhaustive and that other behaviors can easily be added to the framework.

7.2.1. Look-At Behavior

Before performing any grasping or manipulation tasks, it is often necessary to first direct the gaze towards the target object, i.e. the object becomes the focus of attention. Technically, a virtual human does not need to display such a behavior as all information about graphical objects is already known in data structures representing the scene. Yet, gaze behavior significantly increases the naturalness and believability of a virtual character.

In our framework, we implemented a behavior called `Look-At`, which can be used to fixate an arbitrary object. The virtual humans used in the framework have static eyes which can not be turned to either side. Therefore, the gaze behavior is entirely realized through head movements. The `Look-At` behavior takes the position of the object and calculates the rotation needed to turn the head towards this position. Figure 7.5 illustrates the calculation process.

A rotation can be specified using an axis of rotation and a rotation angle (see Chapter 3). In our specific case, the axis of rotation $\boldsymbol{a}$ and the rotation angle δ can be calculated as follows:

$$\delta = \arccos\left(\frac{\boldsymbol{t}\cdot\boldsymbol{d}}{|\boldsymbol{t}\cdot\boldsymbol{d}|}\right) \tag{7.1}$$

$$\boldsymbol{a} = \boldsymbol{t}\times\boldsymbol{d} \tag{7.2}$$

where $\boldsymbol{t}$ is the vector pointing from the center of the head towards the target object and $\boldsymbol{d}$ is the vector pointing in the current direction of the head. Axis and angle form a so-called axis-angle representation of the rotation. The head can be turned to the desired orientation by applying this rotation. For a smooth animation, the `Look-At` performs several interpolation steps between the current direction of the head and the new, calculated direction. In order to achieve higher realism, the interpolation is performed in a slightly nonlinear way using sigmoid functions.

7.2.2. Follow Trajectory Behavior

Trajectories play an important role in the animation of virtual humans and characters. In action capture, for example, the VR user's wrist positions are recorded as trajectories. During animation synthesis, the trajectories can be used to reproduce the humans motions. For this, an inverse kinematics algorithm is applied, such that the wrist of the virtual human moves along the same trajectory in space. This functionality is realized by the `FollowTrajectory` behavior. The behavior takes an arbitrary trajectory in the virtual space as input. Using an inverse kinematics algorithm, it then changes the posture of the virtual human such that the wrist is relocated to the currently active point on the trajectory. Typically, the animation is started at the first point specified and then iterates through all points, until the final point on the trajectory is reached.

In order to ensure reactivity as well as realism, the `FollowTrajectory` behavior also addresses the two challenges of imitation, namely the problem of *adaptation* and *variation*. More specifically, the behavior combines an algorithm for retargeting a given trajectory to a new start- and end-position, together with an algorithm for synthesizing new trajectories. The retargeting algorithm is essential for realizing goal-directed movements, such as reaching, pointing or manipulations of an object. The synthesis algorithm is vital for introducing slight variations to the trajectories. This ensures that trajectories always look a little bit different, even if the same manipulation is repeatedly performed. As a result, the movements of the virtual human appear more lifelike.

The basic idea of the retargeting and synthesis algorithms is the use of a dynamic coordinate system spanned between the hand of the virtual human and the position of the target. The idea is based on behavioral and neurophysiolgical findings which suggest that humans make use of different coordinate systems (CS) for planning and executing

goal-directed behaviors, such as reaching for an object [HS98]. Although the nature of such CS transformations is not yet fully understood, there is empirical support for the critical role of eye-centered, shoulder-centered and hand-centered CS. These are used for transforming a sensory stimulus into motor commands (visuomotor transformations). For retargeting, we use a hand-centered CS which is oriented towards the target object. The basis of this CS is denoted by $B(\boldsymbol{h}, \boldsymbol{o}, \boldsymbol{up})$, where $\boldsymbol{h}$ is the hand position at the beginning of the animation, $\boldsymbol{o}$ is the target (object) position, and $\boldsymbol{up}$ is an up-vector (typically $(0,0,1)^T$).

$$B(\boldsymbol{h}, \boldsymbol{o}, \boldsymbol{up}) = \begin{bmatrix} | & | & | \\ \boldsymbol{x} & \boldsymbol{y} & \boldsymbol{z} \\ | & | & | \end{bmatrix} \quad \text{with} \tag{7.3}$$

$$\boldsymbol{y} = \boldsymbol{o} - \boldsymbol{h} \tag{7.4}$$

$$\boldsymbol{x} = \frac{\boldsymbol{y} \times \boldsymbol{up}}{|\boldsymbol{y} \times \boldsymbol{up}|} \tag{7.5}$$

$$\boldsymbol{z} = \frac{\boldsymbol{x} \times \boldsymbol{y}}{|\boldsymbol{x} \times \boldsymbol{y}|} \tag{7.6}$$

To do so we compute for each trajectory a matrix $T_j^{to\,local}$ which transforms the trajectory into a local space. The origin of this space is centered at the hand position, while the target position is located at $(0,1,0)^T$ on the y-axis.

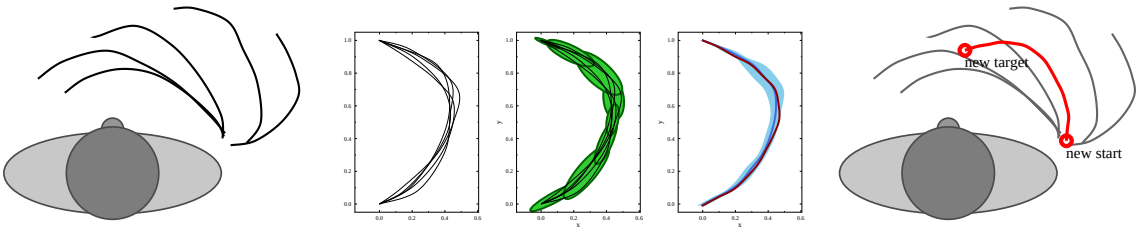
$$\mathsf{T}^{to\,local} = \begin{bmatrix} B(\boldsymbol{x}(0), \boldsymbol{x}(n), \boldsymbol{up})^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & -\boldsymbol{x}(0) \\ 0 & 1 \end{bmatrix} \tag{7.7}$$

$$\mathsf{T}^{to\,global} = \begin{bmatrix} I & \boldsymbol{h}\,' \\ 0 & 1 \end{bmatrix} \begin{bmatrix} B(\boldsymbol{h}\,', \boldsymbol{o}\,', \boldsymbol{up}) & 0 \\ 0 & 1 \end{bmatrix} \tag{7.8}$$

In Figure 7.6 we see the effect of transforming all trajectories into a hand-object CS. The variance, which is due to different goal positions of the reach motion, was removed and the projected trajectories have higher similarity. The new space can be regarded as an end-position invariant space of trajectories.

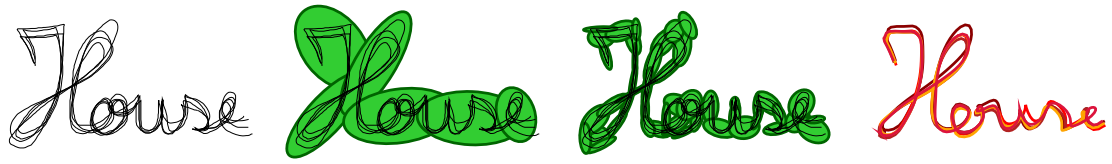
Next, a statistical model of the trajectories is learned. This is done using Gaussian mixture regression (GMR) [CGB07]. The learned GMR model can be queried for a new trajectory having a similar shape to the training trajectories. For a better understanding of the process see Figure 7.7.

The figure shows an example of applying the trajectory synthesis algorithm to handwritten data. In an experiment, a human subject was asked to write the word house several times (a). Using the GMM algorithm, several Gaussians were fitted to the data. The algorithm used 49 Gaussians to optimally enclose the input data (c). Then, using GMR we can sample an arbitrary number of variants of the handwritten word (d).



(a) (b) (c) (d) (e)

Figure 7.6: (a) Trajectories in global-space recorded from a human test-subject. (b) Trajectories in local-space after coordinate system transformation. (c) A GMM is learned by fitting a set of Gaussians. (d) A GMR is learned and new trajectory is synthesized (red). (e) The synthesized trajectory is retargeted based on the new start- and end-position.



(a) (b) (c) (d)

Figure 7.7: Example for the synthesis of handwritten text from examples: (a) Several examples of the word "house" written by a test-subject. (b) A GMM with 6 Gaussians learned from the data. (c) A GMM with 49 Gaussians learned from the data. (d) Several new variations of the word 'house' synthesized by using the proposed approach. The variability follows the learned model.

Once a new trajectory is synthesized, we can retarget it to a new start- and end-position $\boldsymbol{h}'$ and $\boldsymbol{o}'$ by computing $\mathsf{T}^{to\,global}$ and multiplying the resulting matrix with all trajectory points. The complete sequence of steps can be found in Algorithm 5. This algorithm can be used to synthesize a large number of slightly different trajectories, which at the same time share a common structure. The changes in the trajectory avoid repetitiveness and, hence, increase the naturalness of the animation. In addition, the user can specify arbitrary start and ending positions for the trajectories to adapt to a changed environment.

7.2.3. Pick Behavior

The first step to most manipulation tasks is to reach for an object and grasp it. In the Virtual Human Framework the animation of such a movement is realized by the `Pick` behavior which synthesizes animations for picking objects of arbitrary shape and with arbitrary positions and orientations. The user only needs to specify the name of the

Algorithm 5 Synthesizes a random trajectory from given examples and retargets the result to a new start and end-position. A trajectory $\mathcal{T}$ is a set of points $\{\boldsymbol{x}(0), .., \boldsymbol{x}(N)\}$

1: **for** *all* $\mathcal{T} \in Trajectories$ **do**
2: **for** *all* $\boldsymbol{x}(i) \in \mathcal{T}$ **do**
3: $\begin{bmatrix} \boldsymbol{x}(i) \\ 1 \end{bmatrix} \leftarrow \mathsf{T}^{to\,local} \begin{bmatrix} \boldsymbol{x}(i) \\ 1 \end{bmatrix}$
4: **end for**
5: **end for**
6: {*Synthesize new trajectory.*}
7: compute GMM
8: compute new trajectory $\mathcal{T}_{new}$ using GMR
9: {*Ensure trajectory passes through start and end-position.*}
10: **if** $|\boldsymbol{x}(0)| > \epsilon$ **or** $|\boldsymbol{x}(N) - (0/1/0)\,| > \epsilon$ **then**
11: **for** *all* $\boldsymbol{x}(i) \in \mathcal{T}_{new}$ **do**
12: $\begin{bmatrix} \boldsymbol{x}(i) \\ 1 \end{bmatrix} \leftarrow \mathsf{T}^{\,to\,local} \begin{bmatrix} \boldsymbol{x}(i) \\ 1 \end{bmatrix}$
13: **end for**
14: **end if**
15: {*Retarget.*}
16: **for** *all* $\boldsymbol{x}(i) \in \mathcal{T}_{new}$ **do**
17: $\begin{bmatrix} \boldsymbol{x}(i) \\ 1 \end{bmatrix} \leftarrow \mathsf{T}^{\,to\,global} \begin{bmatrix} \boldsymbol{x}(i) \\ 1 \end{bmatrix}$
18: **end for**

object to be grasped, as well as the desired grasp type. Of course, if desired, the user can also exert more control on the parameters of the behavior. For example, he can explicitly specify the direction of approach of the hand during the grasping motion.

To realize the grasping strategy the `Pick` behavior uses a combination of the PLDPM-based grasp synthesis algorithm from Chapter 5 and the `FollowTrajectory` behavior from Section 7.2.2. The synthesis of a new grasping animation consits of two steps, as can be seen in Figure 7.8:

1. **Grasp synthesis** - A stable and natural-looking grasp is generated. Both the intrinsic parameters of the hand (shape), as well as the extrinsic parameters (wrist position, wrist orientation) are optimized. The PLDPMs used for optimization are learned from the data stored in the interaction database.

2. **Trajectory retargeting** - A trajectory recorded during the VR user's interaction is retargeted to the new start and goal position of the animation. The start position corresponds to the current wrist location of the virtual human. The goal position is the optimized wrist position from step 1.

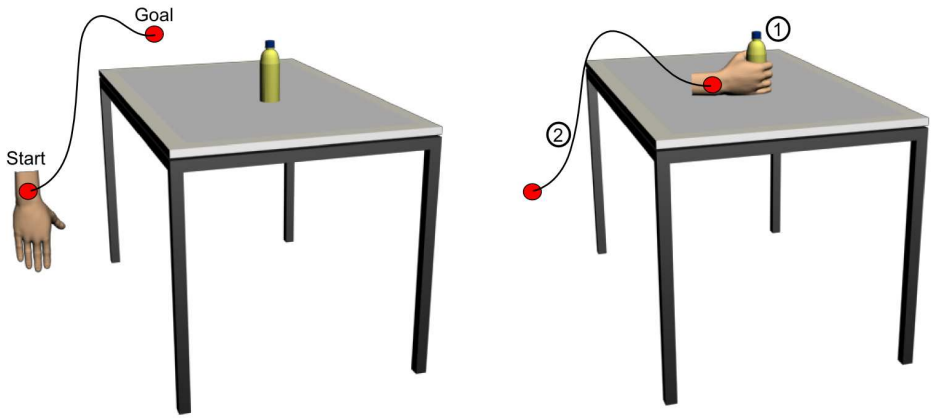


Figure 7.8: Left: The starting point of the algorithm. The goal is to grasp the object by moving along a recorded trajectory. Right: ① First a suitable grasp is found using optimization. ② Then, the new position of the wrist is used to retarget the trajectory.

By performing these two steps, the `Pick` behavior allows the virtual human to realistically grasp arbitrary 3D objects in its surroundings. The optimization process guarantees that the synthesized grasp fits the target object. The retargeting process further ensures that the arm movements of the virtual human match the actual situation. The `Pick` behavior also allows the user to specify a preshape hand configuration. The preshape is a hand posture, that is adopted as the wrist moves towards the object. Typically, the preshape corresponds to an open hand posture, that allows the hand to be closed at the time of grasping. Figure 7.9 shows an animation generated by the `Pick` behavior.

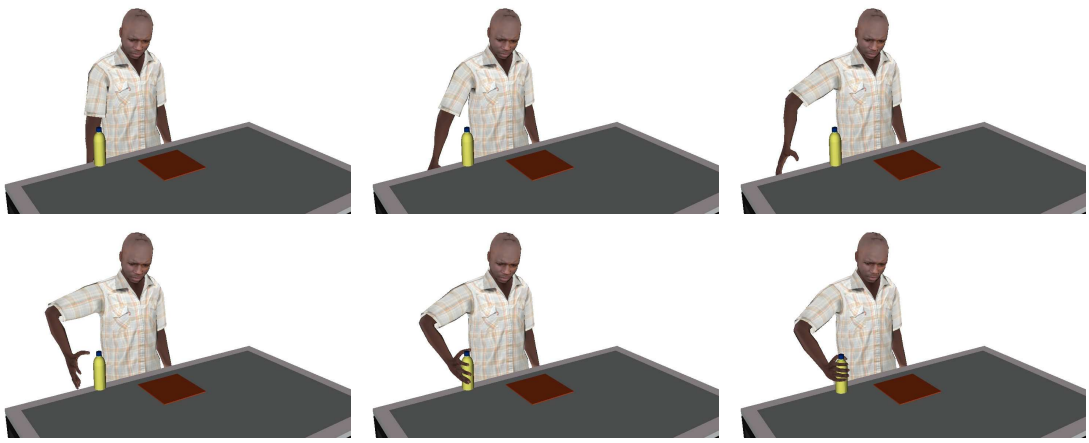


Figure 7.9: A grasping animation generated by the `Pick` behavior. The virtual human grasps a bottle standing on the table.

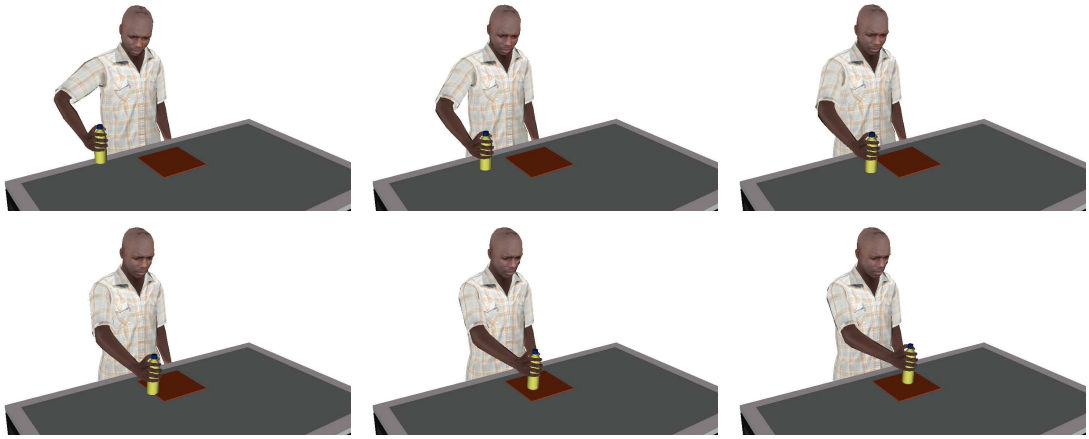


Figure 7.10: Visualization of a `Place` behavior. The virtual human moves the grasped bottle to a specified goal position.

7.2.4. Place Behavior

The `Place` behavior is used to move an object, after grasping to a new position. Hence, `Place` is generally invoked after a `Pick` behavior. The combination of both behaviors can be used to generate animations in which the virtual human picks up a target object and places it at a given position.

The `Place` behavior is based on the functionality provided by the `FollowTrajectory` behavior. The animation is synthesized by retargeting a recorded trajectory which is transformed in a way that it starts at the current position of the hand and finishes at the desired goal position. Figure 7.10 shows an example of this behavior which can be seen as a continuation of the example shown in Figure 7.9. After `Pick` finishes, the virtual human has a firm grasp on the bottle. The `Place` behavior enables the virtual human to move the bottle towards the goal position in the middle of the square spot. Throughout the animation, the behavior also ensures that the object stays attached to the hand.

7.2.5. Relax Behavior

The `Relax` behavior enables the virtual human to take on a relaxed body posture. This is useful for chaining several behaviors together. For example, after performing a `Place` behavior, it can be useful to first relax the body posture of the virtual human before continuing with other tasks. As with the `LookAt` behavior, this improves the visual quality of the animation as well as the believability of the virtual human. The `Relax` behavior generally is also used at the beginning and ending of a behavior sequence. This guarantees that the virtual human always starts and ends the animation with the same relaxed posture. Figure 7.11 shows an example of the `Relax` behavior.

Figure 7.11: Visualization of a `Relax` behavior.

The virtual human starts from its current posture, which typically is the result of behaviors applied prior to the execution of the `Relax` behavior. `Relax` then smoothly interpolates between the current posture and an ideal relaxation posture, in which the head of the virtual human is oriented to the front. The ideal position of the hands is computed by calculating for the left and right hand a position left of the left hip and right of the right hip, respectively. Given these resting positions, the joint configuration of the shoulder, elbow and wrist can be calculated using an inverse kinematics algorithm. The joint configuration of the fingers is derived by using a PLDPM for the cylindrical grasp (see Schlesinger taxonomy in Section 5.2.1).

Once the final configuration of head, arms, and hands is calculated, a spherical linear interpolation is used to smoothly interpolate from the current posture of the virtual human to the calculated relaxation posture.

7.2.6. Push Behavior

A common type of interactions in virtual prototyping scenarios is the pushing of control actuators, buttons or objects. For example, in the earlier introduced espresso machine setting, a button is pushed to start the brewing process of the machine. In the Virtual Human Framework, such interactions are realized through the `Push` behavior. The behavior can simply be parameterized with the target object, as well as the direction, and the stretch of the pushing movement. Figure 7.12 shows an example animation

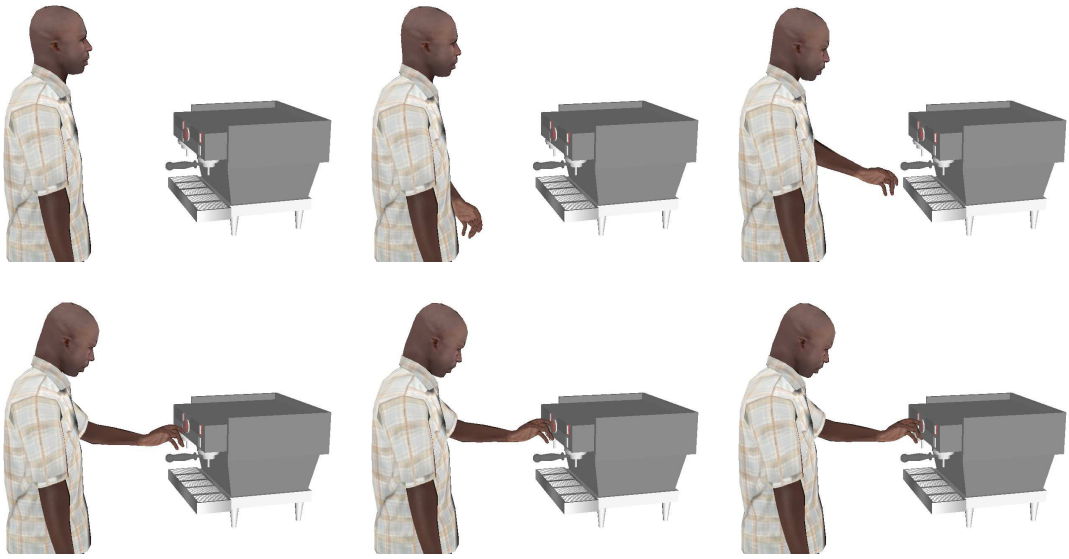


Figure 7.12: Visualization of a `Push` behavior. The virtual human pushes a button.

resulting from a `Push` behavior: the virtual human lifts its hand and arm and then pushes the button on the virtual espresso machine.

The synthesis of this animation requires information about the position of the wrist, as well as the shape of the hand during the pushing interaction. The position of the hand can simply be generated by adding a vector in the opposite direction of the pushing movement to the position of the object. On the other hand, the shape of the hand is generated using a PLDPM trained for *'pressing/pushing a button'*. A detailed account of how such a model can be trained and used was already given in the example section of Chapter 4. Given this information, the animation can be generated by interpolating from the current posture of the virtual human to the target posture. The Virtual Human Framework also ensures that the position of the target object is changed in response to the movements performed.

7.2.7. Turn Behavior

Another important ability of virtual humans in prototyping scenarios is the ability to turn objects. Such an ability is needed, for instance, in order to turn a rotary knob of a technical device or in order to manipulate a door handle. This kind of interaction with objects is mimicked by the `Turn` behavior in the Virtual Human Framework. It takes the target object to be manipulated, as well as the axis of rotation and the rotation angle as input parameters. The behavior first generates a grasp to firmly hold the object. This is achieved by using the grasp synthesis algorithm of Chapter 5. Once the object is grasped,

Figure 7.13: Visualization of a `Turn` behavior. The virtual human rotates a grasped book.

the wrist of the virtual human is turned around the axis of rotation specified using the rotation angle supplied by the user. Figure 7.13 shows an example of the virtual human turning a book. The `Turn` behavior can also be parametrized with different grasp types from the Schlesinger taxonomy. For example, a small rotary knob can be turned using a tip grasp.

7.3. Evaluation and Results

In the following we will present a set of examplary animations that are generated from abstract descriptions of actions as explained in Section 7.1. These animations show virtual humans performing manipulations in a virtual kitchen and a virtual car prototype. To execute the manipulations the virtual humans use the behavior repertoire introduced in Section 7.2. The behaviors are used both in sequence and in parallel.

7.3.1. Virtual Espresso Machine Example

The following example is an extended version of the espresso machine example used in the earlier sections. To test, whether our approach can be used to synthesize complex animations with long sequences of actions, we included actions at the beginning and the end of the espresso machine example. Hence, in the extended version of the example the virtual human must first switch on the espresso machine by pressing the power button. Afterwards, it has to pick a cup located on top of the espresso machine and place it correctly under the coffee outlet. Then, the same actions as in the original example need to be performed: a handle must be turned, the button has to be pressed for pouring the coffee, and the cup must be picked up. To increase the complexity and at the same time the naturalness of the example, we specified that the turning of the handle and the pressing of the button are executed in parallel. Figure 7.14 shows the

Figure 7.14: Brewing of a cup of espresso: the virtual humans first need to press the power-button, then pick up the cup located on top of the espresso machine and place it under the handle. Then they turn the handle and press the button for pouring of the coffee. Finally, they pick up the cup. Although both animations are synthesized from the same plan, there is a visible difference in the joint angles during execution resulting from the adaptation to the anatomy of the virtual humans.

result of replaying this action specification using the Virtual Human Framework. The movement plans which generated the animations can be found in Appendix B.1.

We observe that the introduced method successfully synthesizes animations for two different virtual humans. In spite of the significant difference in size and body proportions, the Virtual Human Framework synthesized meaningful and realistic animations. This is reflected by the correct grasps and manipulations performed. For example, when the cup is picked up from the top of the espresso machine, we can see that there is substantial difference in pose between the virtual humans.

In the figure we can also see that parallel manipulations with both hands (turning the handle and pressing the button with the other hand) are successfully generated. Another interesting aspect is the naturalness of the generated grasps. This aspect becomes evident if we analyze the final part of each animation (lifting the cup to the mouth). We can see, that the grasp synthesized for picking up the cup is a natural-looking precision grasp. Only the thumb and the index finger are bent, all other fingers are extended. This is a realistic hand shape for this task, as it closely matches the way how many people grasp the handle of a cup for drinking. Finally, we observe in Figure 7.14 that the `LookAt` behavior plays an important role in increasing the believability of the virtual human. By turning attention to the current target of manipulation, the virtual human evokes the impression of an intelligent *'living'* being.

7.3.2. Virtual Kitchen Environment Example

This example takes place in a virtual kitchen environment. The goal of the virtual human is to perform a set of manipulations typical for cooking scenarios, e.g. the displacement of a frying pan.

More specifically, the virtual human must first lift the lid of a cooking pot and bend forward to look inside. Then, the virtual human has to grasp a frying pan with the left hand and turn it, until it is accessible to the right hand. After that, it must release the left hand in order to lift the frying pan using the right hand. Finally, the virtual human must move the frying pan away from the stove to place it on the cutting board. The full specification of the chain of behaviors in XML is represented in Appendix B.2.

As obvious from the textual description, this sequence of actions is very dynamic and involves several manipulation steps. A particularly challenging aspect is that the frying pan is first manipulated by the left hand until the right hand takes over the manipulation. Without adaptive behaviors the second picking action would try to grasp the frying pan based on its original position and orientation, which is bound to fail. Figure 7.15 shows stills from two synthesized animations generated with different virtual humans.

Both virtual humans successfully carry out the task described. The transition of the manipulations between the left and right hand is also correctly realized. Furthermore, the grasps synthesized for picking the lid of the pot and the frying pan correctly match the task. This example also shows that the introduced behaviors allow for a large range of complex manipulation procedures when chained together. At the same time,

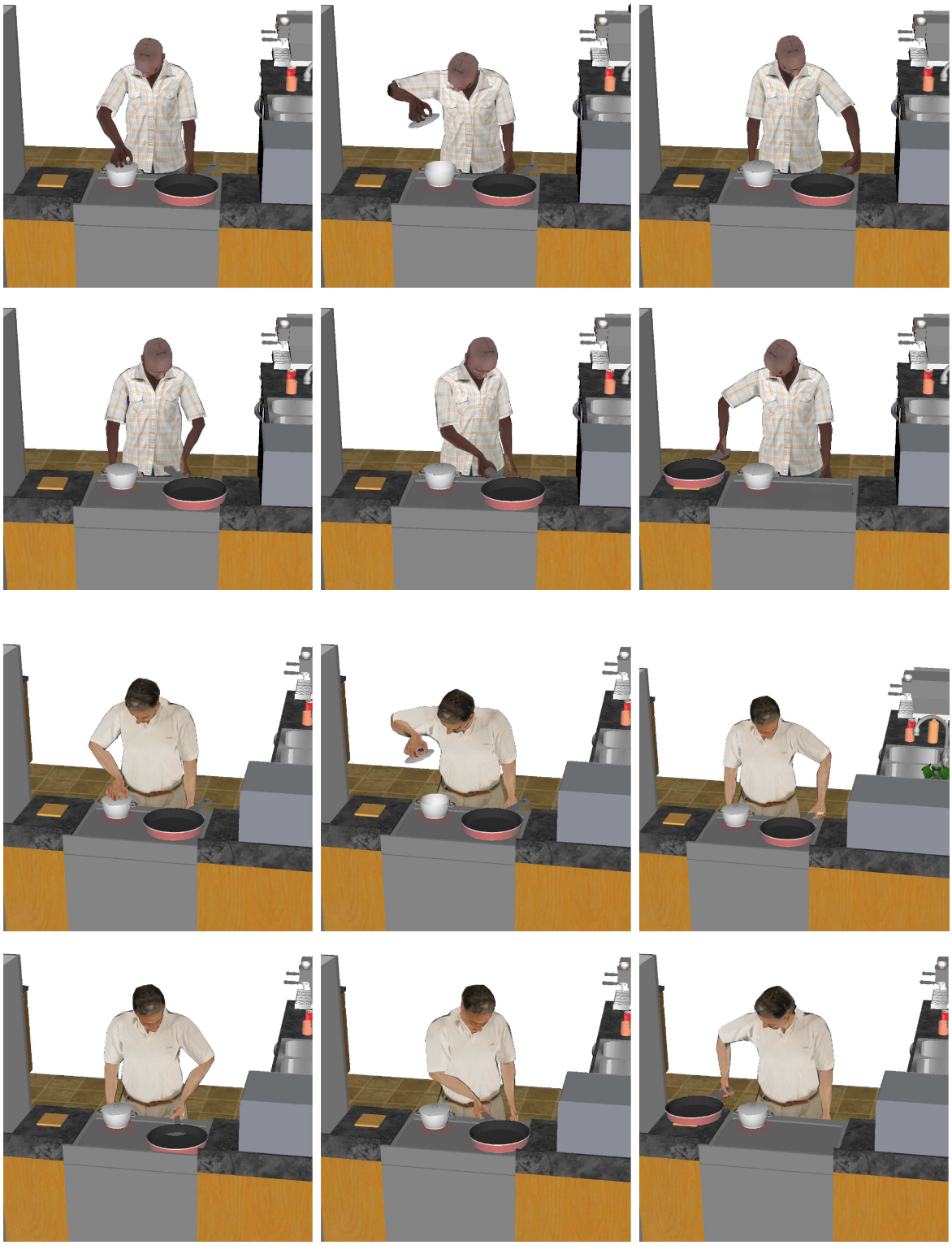


Figure 7.15: Two sequences from synthesized animations in a virtual kitchen environment. The virtual human first lifts the lid of the pot and the turns the frypan until it is accessible to the right hand. Finally, the virtual human moves the frypan to the cutting board. Both animations are synthesized from the same movement plan.

the introduced imitation learning approach using the PLDPM construction algorithm allows us to easily add new motor skills to the behavior repertoire if needed.

7.3.3. Virtual Car Prototype Example

Our final example shows the complete flow of the action capture from the observation to reproduction process. In contrast to the earlier illustrative examples, the movement plan file is not authored, but is derived from a user's interaction in VR. For this, a human demonstrator was asked to perform a set of manipulations on a virtual car prototype. Sitting in front of a Power-Wall the demonstrator is equipped with tracking hardware, including a dataglove and markers. Each of the movements and interactions of the demonstrator with the environment is tracked, recognized and abstracted into actions.

Figure 7.16: Recording session for action capture: the human demonstrator wears a dataglove and tracking markers and performs a set of manipulations in a virtual cockpit of a car. First he grasps the gearshift, then the steering wheel and, finally, the radio button.

Figure 7.16 depicts the situation during the recording session. The user first grasps the gearshift, then the steering wheel and finally the radio button. Each of these interactions triggers one or possibly more interaction events. The interaction events are then automatically translated into actions. For animation these actions are assigned to behaviors, which can be replayed using the Virtual Human Framework. The above recording session yielded a chain of three `Pick` behaviors for grasping the gearshift, steering wheel and the radio button, as well as three `LookAt` behaviors for turning the attention to the respective objects. The full specification of the chain of behaviors in XML is represented in Appendix B.3.

After extracting the behaviors, the Virtual Human Framework is used to synthesize animations for different virtual humans. In Figure 7.17 we see the stills from three animations generated with different virtual humans. Replaying the recorded manipulations using different virtual humans results in various ergonomic insights about the virtual car used for this example. In particular, it shows that the radio button is not easily accessible for various virtual humans. In order to reach for the radio button the virtual

humans have to lean forward. In reality this results in unsafe or dangerous situations. If a driver needs to lean forward to change the volume of the radio, he will be temporarily distracted from driving. Additionally, leaning forward reduces the field of view of the driver.

Figure 7.17: Replay of a recorded action capture session using different virtual humans, which first grasp the gearshift, then the steering wheel and finally the radio button. Some virtual humans have to lean forward to reach the buttons, which can result in unsafe driving situations.

These insights are very helpful for car designers. They can change the car design to improve the security and ergonomy. In our particular example, for instance, a possible solution is to place additional buttons on the steering wheel or to relocate the radio. As a result, the driver can change the volume or the channel without having to lean forwards.

7.4. Conclusion

In this chapter we introduced *action capture*, a new method for learning complex action sequences by imitation. It records and analyzes manipulations performed by a human user in VR. The method generates a multi-layer representation of the recorded actions through a process of abstraction. At the highest level of the abstraction hierarchy is a compact, human-readable representation in an XML-based language. This representation can be used to document, analyze or alter the recorded actions with minimum effort.

We first gave a brief overview of the components of action capture. Then, the chapter mainly focused on the reproduction side of this technique: the question of how to generate new animations from abstract action descriptions. We have shown how actions can be segmented into smaller pieces until they can be replayed based on chains of behaviors. Behaviors are also learned based on the PLDPM construction algorithm discussed in Chapter 4. We discussed a set of behaviors that is derived by this method, in particular behaviors for grasping and manipulation of objects.

In combination with PLDPMs, action capture provides a methodology for imitating both the structure of an action sequence (the correct order and timing of the executed action), as well as details of the particular demonstration (e.g. hand shape or trajectory). Action capture is a VR-based extension of motion capture that takes advantage of interactive virtual environments. Whereas traditional motion capture just aims at replaying the body movements of an actor, action capture further aims at replicating the actor's interactions with the objects of a virtual environment. Valid animations of interactions with scene objects are generated, even if the situation changes. Adaptive behaviors ensure that the synthesized animations match the current environmental context.

The examples in this chapter show how action capture can be used to replay recorded operation procedures using different virtual humans. This is particularly helpful for ergonomic analyzes in virtual prototyping settings. Replaying a recorded operation procedure using virtual humans with different anatomical properties can help to draw conclusions about the design of a technical product. A proof-of-concept was presented in which action capture was used to synthesize manipulations in a virtual car prototype.

8. Conclusion

In this chapter, we first summarize the objectives and results of this thesis in Section 8.1, and then move on to describe our contributions in more detail in Section 8.2. Subsequently in Section 8.3 we present possible future directions and extensions of the presented approach.

8.1. Summary

The aim of this thesis was to propose a new approach for teaching dynamic motor skills to synthetic humanoids, e.g. game characters, avatars or robots. The approach should allow users to easily expand a synthetic humanoid's repertoire of skills. During the conceptualization and implementation of the presented approach, emphasis was put on the following requirements:

- the easy and intuitive specification of new motions
- the ability to adapt and generalize motions to new environments and situations
- the applicability of the approach to different types of synthetic humanoids

To meet the above requirements, we proposed a new type of statistical models named *probabilistic low-dimensional posture model* which is well suited for modeling human postures and motions. The PLDPM construction algorithm exploits the fact that natural motion can often be described as a combination of a small number of components. These components can efficiently be extracted using dimensionality reduction and density estimation techniques, resulting in compact behavior-specific models.

To show how PLDPMs can be used to learn different types of motions from recorded example data, we introduced a three-step imitation learning approach. In contrast to most earlier methods for imitation learning, our approach is not limited to a specific synthetic agent or a particular type of motion. Indeed, we have shown in this thesis that our approach can be successfully used to learn motor skills for virtual humans, small android robots (in simulation and in reality), as well as a state-of-the-art android.

Throughout this thesis we have applied our method to learn a wide range of different motor skills, including behaviors such as *grasping and manipulation of objects*, *standing up*, *walking*, *stair climbing*, *performing a headstand*. In contrast to mere motion capture, the above behaviors are *adaptive*, meaning that they take the current context into

consideration when synthesizing a new motion. This adaptation is realized by searching within a low-dimensional posture space for motion parameters which meet the requirements of the current situation. An optimization algorithm automatically determines the ideal motion parameters, based on a metric of imitation (fitness function) provided by the user. The low-dimensionality of the posture space extracted via a PLDPM renders optimization much more efficient. Further, querying the density function of a PLDPM provides the optimization algorithm with an estimation of the anatomical plausibility of a given posture.

In turn, the optimization process uses the above information to ensure that synthesized motions are faithfully generalized to new situations and environments. As a result, our synthetic humanoids realistically adapt their motions to their surroundings. For example, when imitating a learned walking gait, they adapt their steps to variable terrain or user-specified footprint locations. Optimization was also used to resolve more complex adaptation problems, such as the adjustment of grasps to previously unseen objects, or the adaptation of recorded robot motion to a human interaction partner.

We also applied the presented imitation learning approach to solve important problems in the field of robotics. More specifically, we presented new methods for teaching motor skills to robots by means of intuitive, physical interaction. In this regard, we also addressed the problem of physical human-robot interaction and necessary conditions that need to be met for safe and meaningful interaction and cooperation between humans and robotic systems. To this end we presented *physical interaction learning*, an efficient PLDPM-based learning method for human-in-the-loop learning scenarios. The method was demonstrated using two physical human-robot interaction scenarios. To the best of our knowldege, these experiments are world-wide among the first experiments that feature a learning android robot during close-contact interaction with a human caregiver.

8.2. Contributions

The main contributions of this thesis are summarized as follows:

- **The Probabilistic Low-Dimensional Posture Model** is a compact statistical model of human motion that can be learned from provided example data. The model holds information about the dependecies between different body parts as well as the likelihood of a given posture. Using PLDPMs, a new posture can be specified in terms of a low-dimensional (typically two or three dimensional) vector. Similarly, motions can be specified in terms of low-dimensional trajectories. The synthesis of new postures and motions can, therefore, be cast as an optimization problem in which an optimal point or a curve is searched in the low-dimensional space. We presented a construction algorithm for PLDPM and showed that different dimensionality reduction techniques can be used for this purpose. More specifically, we identified seven dimensionality reduction techniques, ranging from classical linear methods to advanced nonlinear methods, and showed that in particular

principal component analysis, *multidimensional scaling* and *Sammon's nonlinear mapping* are well-suited for extracting compact representations of human motion. Further, building on PLDPMs we have presented a general imitation learning approach which is applicable to different implementations of synthetic humanoids. The approach can be used to learn dynamic motor skills which are always adapted to the current situation by means of optimization. In our experiments we have shown that the approach can be successfully used to learn motor skills at different scales, including finger, hand and full-body motions.

- **A Data-Driven Grasp Synthesis Algorithm** that, given a geometric description of a target object, generates a hand shape that results in a natural-looking and stable grasp. The algorithm uses PLDPMs trained on example motion capture data which was recorded from human subjects. Grasp synthesis is then realized by searching the extracted grasp space for a hand shape that optimizes a given grasp quality metrics. The grasp spaces encoded by the PLDPM can be searched efficiently, while being large enough to contain a variety of plausible grasps. In our experiments, 85% of the grasps generated by the grasp synthesis algorithm were acceptable or better. Further, the mean running time of the algorithm was about three seconds. In contrast to many other grasp synthesis algorithms, as found in robotics literature, our algorithm does not require any pre-processing step in which the object is decomposed or in any other way analyzed.

- **Kinesthetic Bootstrapping** is a new method for programming robots by demonstration. Here, "programming" simply consists of manually moving the robot's joints so as to demonstrate the skill in mind. Throughout the demonstration, the states of the robot's joints are continuously recorded and later used to learn a corresponding PLDPM. The demonstrated skill is then replayed and further optimized within a physics-driven virtual environment. After optimization in the virtual environment the optimized skill is transferred back to the robot and replayed in reality. Kinesthetic bootstrapping allows for an easy and intuitive way of programming robots based on natural touch interactions. Hence, the approach is accessible to non-experts and does not require prior knowledge on robotics. We have demonstrated the efficiency of the method by learning highly dynamic skills, such as walking, standing up, or performing a headstand, using a small humanoid robot.

- **Physical Interaction Learning** improves the cooperation of humans and robots *while* they are working to achieve a common goal. The human interaction partner can give feedback about the success of the interaction to the robot. The robot then collects information about recent successful interactions and learns a new PLDPM-based control policy. The learned control policy then drives his motions in the next interaction step. Physical interaction learning targets new application domains in which robots and humans have to work together to achieve a task.

Interaction and cooperation between humans and robots becomes an increasingly important and, at the same time, challenging aspect of robot development. In an experiment inspired by parenting behavior in humans, we were able to show that the proposed learning method results in measurable improvements of the interaction. Quantitative evaluations based on the posture change norm confirm the significance of these improvements.

- **A Framework for the Imitation of Action Sequences** was presented which builds upon the action capture [JAHV10] technique. The framework allows us to synthesize long sequences of actions from textual descriptions. These textual descriptions can be either hand-crafted or recorded from human interactions in an immersive virtual environment. The actions are mapped onto behaviors learned using PLDPMs and the presented imitation learning approach. Once an action sequence is specified, it can be replayed using a large number of different virtual humans.

8.3. Future Directions

8.3.1. Application to Robotics

In order to gather information about the environment, modern robots often use a range of different sensor technologies. This can include acceleration sensors, cameras, or tactile sensors. Especially interesting for our purposes are tactile sensors as they can provide us with important information about physical interactions between a human and a robot. For example, in the physical interaction learning scenario presented in this thesis, touch sensors could give us information about the amount of force exerted by the human on the robot's skin as well as the location of the touch interaction. In turn, this information could be used to replace the human critique used so far in our method. As a result, we would have a more finegrained criterion on which to base the optimization process. In contrast to the binary critique employed so far, touch sensors provide us with continuous measurements for guiding the learning process. Indeed, the CB^2 robot used in this thesis is already equiped with malleable tactile sensors which cover the whole body. In our experiments, however, we found that the signals generated by these sensors are not reliable. This is due to the seamless skin of the robot. With every movement of the robot the skin is also deformed, causing tactile sensations without any touch interaction taking place. Therfore, to overcome this problem we need methods for discriminating between tactile sensations caused by the interaction with the external environment and tactile sensations originating from the robot's own movements. Interesting early approaches for discriminating self from others have been presented in [MYN+07].

Thus far, the low-level control system used to actuate our robots consisted of simple proportional-derivative (PD) controllers. Given the desired joint configuration of the

robot, PD controllers computed the needed torques and forces using a feedback mechanism. For a good performance, parameter tuning of the *proportional* and *derivative* terms of these controllers is needed. However, when dealing with complex robotic systems it is often not efficient to specify constant values for these parmeters. In order to achieve high precision control the parameters need to be changed over time instead. To avoid these problems, more intelligent and adaptive low-level control systems can be used. In our work in [BIT+07] we proposed an intelligent control system based on neural networks which automatically learns ideal parameters for achieving high-precision control of the robot.

8.3.2. Application to Computer Animation

The proposed imitation learning approach has wide application perspectives in the game and movie industry. To this day, these domains mainly employ keyframe and motion capture techniques for specifying new motions of virtual humans, i.e. techniques that do not support the adaptation to new situations. In contrast to that, our approach is well suited for this kind of task. The methods and algorithms presented in this thesis can be easily integrated into modern computer animation software, such as 3D Studio Max, Motion Builder, or Maya. The presented grasp synthesis algorithm, for instance, can help the animator to easily fit a character's handshape to a modeled 3D object. The techniques in Chapter 7 can also be used in these software tools to specify animations by means of a compact action description instead of time-consuming keyframes.

While a potential application in the movie industry is straight-forward, there are important aspects that need to be addressed before successful application in the gaming industry. Specifically, the synthesis process must be made more efficient, in order to meet the strong real-time requirements of games. Typically modern games assign the largest part of the available processing power to rendering and graphics calculations. Other parts of the game code, such as AI or sound calculations have less processing power assigned. Therefore, algorithms that deal with non-graphical aspects of the game need to be highly efficient and computationally cheap.

For example, while the presented grasp synthesis algorithm is significantly faster than comparable methods presented in the literature (e.g. [LFP07]), it is still too slow for game applications. The detailed analysis of this and the other motion synthesis examples showed that the main computational bottleneck is the calculation of distances between objects. The synthesis times can be drastically reduced if the computation times for distances can be reduced. One way to do this is to employ more sophisticated proximity query algorithms. An even more promising approach is to perform the distance calculation on the graphics hardware. For instance in [SOM04], Sud and colleagues proposed a fast algorithm for distance field computation on graphics hardware. According to the authors, the running time of proximity computations was improved by an order of magnitude in comparison to earlier state-of-the-art techniques. It is therefore reasonable to assume that the replacement of our distance calculation code by the

algorithm of Sud et al. can lead to a significant performance improvement of the motion synthesis process. Similarly, the evolutionary algorithm underlying the synthesis process can also be executed on the graphics hardware for faster computation times.

Another interesting field of future work is the integration of path-planning techniques. So far, our retargeting method can adapt trajectories to new start- and end-positions. However, there might also be an obstacle blocking the way between the two positions. To realistically react to such a situation, a virtual human needs to modify the shape of the trajectory such that the obstacle is avoided. Several path-planning techniques have been proposed in the literature which could be adopted for this purpose (see for example [Lat91] for a classical textbook on the topic, or [VBA+09] for a modern path-planning approach).

8.4. Concluding Remarks

This thesis addressed the question of how to teach dynamic motor skills to synthetic humanoids. A general approach based on imitation learning was presented and evaluated on a number of synthetic humanoids, as well as a number of different motor skills. Our approach allows for intuitive and natural specification of motor skills without the need for expert knowledge. Using this approach we showed that various important problems in robotics and computer animation can be tackled, including the synthesis of natural grasping, the synthesis of locomotion behavior or the physical interaction between humans and robots. The application domains for the techniques presented in this thesis are therefore manifold.

A. Virtual Objects

Figure A.1 shows all objects that have been used for the grasp synthesis experiment in Chapter 5. Each 3D object is modelled after a real, physical object which was used for recording the training data.

Figure A.1: Virtual objects used for grasp synthesis experiments (from left to right): bottle, coffe jug, table, screwdriver, mouse, CD, matchbox, key, taperoller, bag, rubber, tennisball, Stanford bunny, toy, nail, pencil, hammer, joystick, case, suitcase, card.

B. Example Behavior Plans

B.1. Virtual Espresso Machine Example

```
<?xml version="1.0" ?>
<movement-plans>
  <default-parameters>
    <param name="version">3</param>
    <param name="boneNamingType">hanim</param>
  </default-parameters>
  <plan>
    <name>Pouring Coffee</name>
    <behavior>
      <type>Relax</type>
      <param name="start-time">0</param>
      <param name="end-time">1</param>
      <param name="relax-overswing">0.05</param>
    </behavior>
    <behavior>
      <type>LookAt</type>
      <param name="start-time">1</param>
      <param name="end-time">1.7</param>
      <param name="side">none</param>
      <param name="lookat-object">button_coffee_middle</param>
      <param name="lookat-distribution">1</param>
    </behavior>
    <behavior>
      <type>Push</type>
      <param name="start-time">1</param>
      <param name="end-time">1.8</param>
      <param name="side">left</param>
      <param name="object">button_coffee_middle</param>
      <param name="grasp-type">extension</param>
      <param name="grasp-end-shape">0 0 0</param>
      <param name="follow-ik-method">IkArmHeuristic</param>
      <param name="push-move-direction">-1 0 0</param>
      <param name="push-move-distance">0.01</param>
      <param name="push-distribution">0.8</param>
      <param name="push-optimize">near</param>
    </behavior>
    <behavior>
      <type>Push</type>
      <param name="start-time">1.8</param>
      <param name="end-time">2</param>
      <param name="side">left</param>
      <param name="object">button_coffee_middle</param>
      <param name="grasp-type">extension</param>
      <param name="grasp-end-shape">0 0 0</param>
      <param name="follow-ik-method">IkArmHeuristic</param>
      <param name="push-move-direction">-1 0 0</param>
      <param name="push-move-distance">-0.01</param>
      <param name="push-distribution">0</param>
      <param name="push-optimize">near</param>
    </behavior>
    <behavior>
      <type>Relax</type>
      <param name="start-time">2</param>
      <param name="end-time">3</param>
      <param name="relax-overswing">0.05</param>
      <param name="relax-head">0</param>
      <param name="relax-arm-right">0</param>
      <param name="relax-hand-right">0</param>
    </behavior>
    <behavior>
      <type>LookAt</type>
      <param name="start-time">1.8</param>
```

```
    <param name="end-time">4.5</param>
    <param name="side">none</param>
    <param name="lookat-object">espresso_cup</param>
    <param name="lookat-distribution">0.25</param>
  </behavior>
  <behavior>
    <type>Pick</type>
    <param name="start-time">1.8</param>
    <param name="end-time">3.25</param>
    <param name="side">right</param>
    <param name="object">espresso_cup</param>
    <param name="grasp-type">Schlesinger::spherical</param>
    <param name="grasp-object-interaction">attach</param>
    <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
    <param name="follow-ik-method">IkArmHeuristic</param>
    <param name="pick-results-file">results/demoKitchenPlan05einfach-1.xml</param>
  </behavior>
  <behavior>
    <type>Place</type>
    <param name="start-time">3.25</param>
    <param name="end-time">4.5</param>
    <param name="side">right</param>
    <param name="object">espresso_cup</param>
    <param name="follow-trajectory-points">
      0    0.0    0.0    0.2
      2    0.0    0.0    0.25
      3    0.25    0.0    0.25
      5    0.2    0.0    0.0
      7    0.0    0.0    0.0
    </param>
    <param name="follow-use-bowing">1</param>
    <param name="follow-ik-method">IkArmHeuristic</param>
    <param name="wrist-use-global-rot">1</param>
    <param name="place-obj-pos">0.66 0.52 1.05</param>
    <param name="place-obj-rot">0 0 0 1</param>
  </behavior>
  <behavior>
    <type>Grasp</type>
    <param name="start-time">4.5</param>
    <param name="end-time">5</param>
    <param name="side">right</param>
    <param name="object">espresso_cup</param>
    <param name="grasp-type">Schlesinger::spherical</param>
    <param name="grasp-pre-shape">-0 0 0</param>
    <param name="grasp-end-shape">-1 0 0</param>
    <param name="grasp-object-interaction">detach</param>
  </behavior>
  <behavior>
    <type>Relax</type>
    <param name="start-time">5</param>
    <param name="end-time">6</param>
    <param name="relax-overswing">0.05</param>
    <param name="relax-head">0</param>
    <param name="relax-arm-left">0</param>
    <param name="relax-hand-left">0</param>
  </behavior>
  <behavior>
    <type>LookAt</type>
    <param name="start-time">4.5</param>
    <param name="end-time">5.5</param>
    <param name="side">none</param>
    <param name="lookat-object">control_left</param>
    <param name="lookat-distribution">1</param>
  </behavior>
  <behavior>
    <type>Bowing</type>
    <param name="start-time">4.5</param>
    <param name="end-time">6</param>
    <param name="side">none</param>
    <param name="bowing-bone">vt1 10 -1 0 0</param>
    <param name="bowing-bone">vt6 10 -1 0 0</param>
    <param name="bowing-bone">vt10 10 -1 0 0</param>
  </behavior>
  <behavior>
    <type>Pick</type>
    <param name="start-time">4.5</param>
    <param name="end-time">6</param>
    <param name="side">left</param>
    <param name="object">control_left</param>
    <param name="grasp-type">Schlesinger::cylindrical</param>
    <param name="grasp-object-interaction">detach</param>
    <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
    <param name="follow-ik-method">IkArmHeuristic</param>
    <param name="pick-results-file">results/demoKitchenPlan05einfach-2.xml</param>
  </behavior>
```

```
<behavior>
  <type>Turn</type>
  <param name="start-time">6</param>
  <param name="end-time">6.5</param>
  <param name="side">left</param>
  <param name="object">control_left</param>
  <param name="turn-origin">-0.1 0.0 0</param>
  <param name="turn-axis">0 0 1</param>
  <param name="turn-angle">20</param>
  <param name="follow-ik-method">IkArmHeuristic</param>
</behavior>
<behavior>
  <type>Bowing</type>
  <param name="start-time">9.25</param>
  <param name="end-time">10.5</param>
  <param name="side">none</param>
  <param name="bowing-bone">vt1 0 1 0 0</param>
  <param name="bowing-bone">vt6 0 1 0 0</param>
  <param name="bowing-bone">vt10 0 1 0 0</param>
</behavior>
<behavior>
  <type>LookAt</type>
  <param name="start-time">5.5</param>
  <param name="end-time">6</param>
  <param name="side">none</param>
  <param name="lookat-object">button_coffee_white_right</param>
  <param name="lookat-distribution">1</param>
</behavior>
<behavior>
  <type>Push</type>
  <param name="start-time">6</param>
  <param name="end-time">7</param>
  <param name="side">right</param>
  <param name="object">button_coffee_white_right</param>
  <param name="grasp-type">extension</param>
  <param name="grasp-end-shape">-1 0 0</param>
  <param name="follow-ik-method">IkArmHeuristic</param>
  <param name="push-move-direction">-1 0 0</param>
  <param name="push-move-distance">0.0025</param>
  <param name="push-distribution">0.75</param>
  <param name="push-optimize">near</param>
</behavior>
<behavior>
  <type>LookAt</type>
  <param name="start-time">7</param>
  <param name="end-time">11.4</param>
  <param name="side">none</param>
  <param name="lookat-object">espresso_cup</param>
  <param name="lookat-distribution">0.075</param>
</behavior>
<behavior>
  <type>Push</type>
  <param name="start-time">8.75</param>
  <param name="end-time">9</param>
  <param name="side">right</param>
  <param name="object">button_coffee_white_right</param>
  <param name="grasp-type">extension</param>
  <param name="grasp-end-shape">-1 0 0</param>
  <param name="follow-ik-method">IkArmHeuristic</param>
  <param name="push-move-direction">-1 0 0</param>
  <param name="push-move-distance">-0.0025</param>
  <param name="push-distribution">0</param>
  <param name="push-optimize">near</param>
</behavior>
<behavior>
  <type>Relax</type>
  <param name="start-time">9</param>
  <param name="end-time">10</param>
  <param name="relax-overswing">0.05</param>
  <param name="relax-head">0</param>
  <param name="relax-arm-left">0</param>
  <param name="relax-hand-left">0</param>
</behavior>
<behavior>
  <type>Pick</type>
  <param name="start-time">9.25</param>
  <param name="end-time">10.5</param>
  <param name="side">left</param>
  <param name="object">espresso_cup</param>
  <param name="grasp-type">Schlesinger::tip</param>
  <param name="grasp-object-interaction">attach</param>
  <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
  <param name="follow-ik-method">IkArmHeuristic</param>
  <param name="pick-results-file">results/demoKitchenPlan05einfach-3.xml</param>
</behavior>
```

```
    <behavior>
      <type>Place</type>
      <param name="start-time">10.5</param>
      <param name="end-time">11.75</param>
      <param name="side">left</param>
      <param name="object">espresso_cup</param>
      <param name="follow-trajectory-points">
        0    0.05     0.2    -0.35
        1    0.05     0.05     -0.35
        2    0.0    0.05     0.0
        3    0.0    0.0    0.0
      </param>
      <param name="follow-use-bowing">0</param>
      <param name="follow-ik-method">IkArmHeuristic</param>
      <param name="wrist-use-global-rot">1</param>
      <param name="place-obj-pos">0.88 0.51 1.62</param>
      <param name="place-obj-rot">45 0 1 0</param>
    </behavior>
    <behavior>
      <type>LookAt</type>
      <param name="start-time">12.75</param>
      <param name="end-time">14</param>
      <param name="side">none</param>
      <param name="lookat-object">espresso_cup</param>
      <param name="lookat-distribution">0.5</param>
    </behavior>
    <behavior>
      <type>Place</type>
      <param name="start-time">12.75</param>
      <param name="end-time">13.75</param>
      <param name="side">left</param>
      <param name="object">espresso_cup</param>
      <param name="follow-trajectory-points">
        0    0.0    0.0    0.0
        1    1.0    1.0    1.0
        2    2.0    2.0    2.0
      </param>
      <param name="follow-use-bowing">0</param>
      <param name="follow-ik-method">IkArmHeuristic</param>
      <param name="wrist-use-global-rot">1</param>
      <param name="place-obj-pos">0.669981 0.232822 1.34276</param>
      <param name="place-obj-rot">0 0 1 0</param>
    </behavior>
    <behavior>
      <type>Grasp</type>
      <param name="start-time">13.75</param>
      <param name="end-time">14</param>
      <param name="side">left</param>
      <param name="object">espresso_cup</param>
      <param name="grasp-type">Schlesinger::cylindrical</param>
      <param name="grasp-pre-shape">0 0 0</param>
      <param name="grasp-end-shape">-1 0 0</param>
      <param name="grasp-object-interaction">detach</param>
    </behavior>
    <behavior>
      <type>Relax</type>
      <param name="start-time">14</param>
      <param name="end-time">15</param>
      <param name="relax-overswing">0.05</param>
    </behavior>
  </plan>
</movement-plans>
```

Listing B.1: An XML representation of the chain of behaviors needed for brewing a cup of espresso.

B.2. Virtual Kitchen Environment Example

```
<?xml version="1.0" ?>
<movement-plans>
  <default-parameters>
    <param name="version">3</param>
    <param name="boneNamingType">hanim</param>
  </default-parameters>
  <plan>
    <name>Cooking with Frying Pan</name>
    <behavior>
      <type>Relax</type>
      <param name="start-time">0</param>
      <param name="end-time">1</param>
      <param name="relax-overswing">0.05</param>
    </behavior>
    <behavior>
      <type>LookAt</type>
      <param name="start-time">2.0</param>
      <param name="end-time">5.5</param>
      <param name="lookat-object">pot</param>
    </behavior>
    <behavior>
      <type>Pick</type>
      <param name="start-time">2</param>
      <param name="end-time">3</param>
      <param name="side">right</param>
      <param name="object">top</param>
      <param name="grasp-type">Schlesinger::spherical</param>
      <param name="grasp-object-interaction">attach</param>
      <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
      <param name="follow-ik-method">IkArmSimple</param>
      <param name="pick-results-file">results/demoKitchenPlan_herd-1.xml</param>
    </behavior>
    <behavior>
      <type>Bowing</type>
      <param name="start-time">3</param>
      <param name="end-time">4</param>
      <param name="side">left</param>
      <param name="bowing-bone">vt1 5 0 0 1</param>
      <param name="bowing-bone">vt6 5 0 0 1</param>
      <param name="bowing-bone">vt10 10 0 0 1</param>
    </behavior>
    <behavior>
      <type>Place</type>
      <param name="start-time">3</param>
      <param name="end-time">4</param>
      <param name="side">right</param>
      <param name="object">top</param>
      <param name="follow-trajectory-points">
        0.0 0.0 0.0 0.0
        1.0 0.0 0.0 1.0
      </param>
      <param name="follow-use-bowing">0</param>
      <param name="follow-ik-method">IkArmSimple</param>
      <param name="wrist-use-global-rot">1</param>
      <param name="place-obj-pos">1.70 2.75 1.31</param>
      <param name="place-obj-rot">20 0 1 0</param>
    </behavior>
    <behavior>
      <type>Bowing</type>
      <param name="start-time">5</param>
      <param name="end-time">6</param>
      <param name="side">left</param>
      <param name="bowing-bone">vt1 0 0 0 1</param>
      <param name="bowing-bone">vt6 0 0 0 1</param>
      <param name="bowing-bone">vt10 0 0 0 1</param>
    </behavior>
    <behavior>
      <type>Place</type>
      <param name="start-time">5</param>
      <param name="end-time">6</param>
      <param name="side">right</param>
      <param name="object">top</param>
      <param name="follow-trajectory-points">
        0.0 0.0 0.0 0.0
        1.0 0.0 0.0 1.0
      </param>
      <param name="follow-use-bowing">0</param>
      <param name="follow-ik-method">IkArmSimple</param>
```

```
    <param name="wrist-use-global-rot">1</param>
    <param name="place-obj-pos">1.6115  2.75  1.005</param>
    <param name="place-obj-rot">0 0 1 0</param>
  </behavior>
  <behavior>
    <type>LookAt</type>
    <param name="start-time">5.5</param>
    <param name="end-time">11</param>
    <param name="lookat-object">pan</param>
  </behavior>
  <behavior>
    <type>Relax</type>
    <param name="start-time">6</param>
    <param name="end-time">7</param>
    <param name="relax-overswing">0.05</param>
    <param name="relax-arm-left">0</param>
    <param name="relax-hand-left">0</param>
    <param name="relax-head">0</param>
  </behavior>
  <behavior>
    <type>Pick</type>
    <param name="start-time">6</param>
    <param name="end-time">7</param>
    <param name="side">left</param>
    <param name="object">pan</param>
    <param name="grasp-type">Schlesinger::cylindrical</param>
    <param name="grasp-object-interaction">attach</param>
    <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
    <param name="follow-ik-method">IkArmSimple</param>
    <param name="pick-results-file">results/demoKitchenPlan_herd-2.xml</param>
  </behavior>
  <behavior>
    <type>Turn</type>
    <param name="start-time">7</param>
    <param name="end-time">8</param>
    <param name="side">left</param>
    <param name="object">pan</param>
    <param name="turn-origin">0.0 0.0 0.0</param>
    <param name="turn-axis">0.001 0.001 1.0</param>
    <param name="turn-angle">50</param>
    <param name="follow-ik-method">IkArmSimple</param>
  </behavior>
  <behavior>
    <type>Bowing</type>
    <param name="start-time">8</param>
    <param name="end-time">9</param>
    <param name="side">none</param>
    <param name="bowing-bone">vt1 15 1 0 0</param>
    <param name="bowing-bone">vt6 15 1 0 0</param>
    <param name="bowing-bone">vt10 15 1 0 0</param>
  </behavior>
  <behavior>
    <type>Relax</type>
    <param name="start-time">8</param>
    <param name="end-time">9</param>
    <param name="relax-overswing">0.05</param>
    <param name="relax-arm-right">0</param>
    <param name="relax-hand-right">0</param>
    <param name="relax-head">0</param>
  </behavior>
  <behavior>
    <type>Pick</type>
    <param name="start-time">8</param>
    <param name="end-time">9</param>
    <param name="side">right</param>
    <param name="object">pan</param>
    <param name="grasp-type">Schlesinger::cylindrical</param>
    <param name="grasp-object-interaction">attach</param>
    <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
    <param name="follow-ik-method">IkArmSimple</param>
    <param name="wrist-use-global-rot">1</param>
    <param name="pick-results-file">results/demoKitchenPlan_herd-3.xml</param>
  </behavior>
  <behavior>
    <type>Bowing</type>
    <param name="start-time">9</param>
    <param name="end-time">10</param>
    <param name="side">none</param>
    <param name="bowing-bone">vt1 0 1 0 0</param>
    <param name="bowing-bone">vt6 0 1 0 0</param>
    <param name="bowing-bone">vt10 0 1 0 0</param>
  </behavior>
  <behavior>
    <type>Place</type>
    <param name="start-time">9</param>
```

```
      <param name="end-time">10</param>
      <param name="side">right</param>
      <param name="object">pan</param>
      <param name="follow-trajectory-points">
        0.0   0.0   0.0   0.0
        1.0   0.0   0.1   0.3
        2.0   1.0   0.1   0.3
        3.0   1.0   0.0   0.0
      </param>
      <param name="follow-use-bowing">0</param>
      <param name="follow-ik-method">IkArmSimple</param>
      <param name="wrist-use-global-rot">1</param>
      <param name="place-obj-pos">2.1  2.4  0.93</param>
      <param name="place-obj-rot">40 0 0 1</param>
    </behavior>

    <behavior>
      <type>Relax</type>
      <param name="start-time">11</param>
      <param name="end-time">12</param>
      <param name="relax-overswing">0.05</param>
    </behavior>
  </plan>
</movement-plans>
```

Listing B.2: An XML representation of the chain of behaviors needed for performing some cooking manipulations.

B.3. Virtual Car Prototype Example

```
<?xml version="1.0" ?>
<movement-plans>
  <default-parameters>
    <param name="version">3</param>
    <param name="boneNamingType">hanim</param>
  </default-parameters>
  <plan>
    <name>Cockpit Interactions</name>
    <behavior>
      <type>Relax</type>
      <param name="start-time">0</param>
      <param name="end-time">1</param>
      <param name="relax-overswing">0.05</param>
    </behavior>
    <behavior>
      <type>LookAt</type>
      <param name="start-time">0.0</param>
      <param name="end-time">5.0</param>
      <param name="lookat-object">gear_shift</param>
    </behavior>
    <behavior>
      <type>Pick</type>
      <param name="start-time">2</param>
      <param name="end-time">5</param>
      <param name="side">right</param>
      <param name="object">gear_shift</param>
      <param name="grasp-type">Schlesinger::spherical</param>
      <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
      <param name="follow-ik-method">IkArmSimple</param>
    </behavior>
    <behavior>
      <type>LookAt</type>
      <param name="start-time">5</param>
      <param name="end-time">8</param>
      <param name="lookat-object">steering_wheel</param>
    </behavior>
    <behavior>
      <type>Pick</type>
      <param name="start-time">5</param>
      <param name="end-time">8</param>
      <param name="side">right</param>
      <param name="object">steering_wheel</param>
      <param name="grasp-type">Schlesinger::spherical</param>
      <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
      <param name="follow-ik-method">IkArmSimple</param>
    </behavior>
    <behavior>
      <type>LookAt</type>
      <param name="start-time">8</param>
      <param name="end-time">11</param>
      <param name="lookat-object">radio_button</param>
    </behavior>
    <behavior>
      <type>Pick</type>
      <param name="start-time">8</param>
      <param name="end-time">11</param>
      <param name="side">right</param>
      <param name="object">radio_button</param>
      <param name="grasp-type">Schlesinger::spherical</param>
      <param name="follow-trajectory-file">trajectories/Trajectory_1_global_right.trj</param>
      <param name="follow-ik-method">IkArmSimple</param>
    </behavior>
  </plan>
</movement-plans>
```

Listing B.3: An XML representation of the chain of behaviors for performing manipulations in a virtual car prototype.

List of Figures

List of Tables

List of Algorithms

Listings

Bibliography

[AAGD08] T. Asfour, P. Azad, F. Gyarfas, and R. Dillmann. Imitation learning of dual-arm manipulation tasks in humanoid robots. *International Journal of Humanoid Robotics*, 5(2):183–202, 2008.

[ABIO00] M. A. Arbib, A. Billard, M. Iacoboni, and E. Oztop. Synthetic brain imaging: grasping, mirror neurons and imitation. *Neural Networks*, 13(8-9):975 – 997, 2000.

[AFM98] A. Alexandrov, A. Frolov, and J. Massion. Axial synergies during human upper trunk bending. *Exp. Brain Res.*, 118(2), 1998.

[AM00] M. Alexa and W. Müller. Representing animations by principal components. *Comput. Graph. Forum*, 19(3), 2000.

[Arb02] M. A. Arbib. The mirror system, imitation, and the evolution of language. In K. Dautenhahn and C. Nehaniv, editors, *Imitation in Animals and Artefacts*. MIT Press, 2002.

[Arb08] M. A. Arbib. From grasp to language: Embodied concepts and the challenge of abstraction. *Journal of Physiology-Paris*, 102(1-3):4 – 20, 2008. Links and Interactions Between Language and Motor Systems in the Brain.

[ARTG10] Advanced Realtime Tracking GmbH. Homepage. www.ar-tracking.eu, last visited on May 1st 2010., 2010.

[BCDS08] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Survey: Robot programming by demonstration. In *Handbook of Robotics*, volume chapter 59. MIT Press, 2008.

[BdST02] E. Bizzi, A. d'Avella, P. Saltiel, and MC. Tresch. Modular organization of spinal motor systems. *The Neuroscientist*, 8(5), 2002.

[BH00] M. Brand and A. Hertzmann. Style machines. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[BI05] C. W. Borst and A. P. Indugula. Realistic virtual grasping. In *VR '05: Proceedings of the 2005 IEEE Conference on Virtual Reality*, pages 91–98, 2005.

[Bil00] A. Billard. Learning motor skills by imitation: a biologically inspired robotic model. *Cybernetics & Systems*, 32(1-2):155–193, 2000.

[Bis87] R. Bisiani. Beam search. In *Encyclopedia of Artificial Intelligence*, pages 56–58. Wiley & Sons, 1987.

[BIT+07] H. Ben Amor, S. Ikemoto, T.Minato, B. Jung, and H. Ishiguro. A neural framework for robot motor learning based on memory consolidation. In Bartlomiej Beliczynski, Andrzej Dzielinski, Marcin Iwanowski, and Bernardete Ribeiro, editors, *ICANNGA (2)*, volume 4432 of *Lecture Notes in Computer Science*, pages 641–648. Springer, 2007.

[BK96] P. Bakker and Y. Kuniyoshi. Robot see, robot do: An overview of robot imitation. In *AISB96 Workshop: Learning in Robots and Animals*, pages 3–11, 1996.

[BL09] E. I. Barakova and T. Lourens. Mirror neuron framework yields representations for robot interaction. *Neurocomput.*, 72(4-6):895–900, 2009.

[BT03] O. A. Bauchau and L. Trainelli. The vectorial parametrization of rotation. *Nonlinear Dynamics*, 32(1), 2003.

[BTD10] T. Bockemuehl, N. F. Troje, and V. Duerr. Inter-joint coupling and joint angle synergies of human catching movements. *Human Movement Science*, 29(1):73 – 93, 2010.

[BV09] S. Bitzer and S. Vijayakumar. Latent spaces for dynamic movement primitives. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, Dec. 2009.

[BW95] A. Bruderlin and L. Williams. Motion signal processing. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 97–104, New York, NY, USA, 1995. ACM.

[CD88] W. S. Cleveland and S. J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83(403):596–610, 1988.

[CGB07] 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. Special issue on robot learning by observation, demonstration and imitation*, 37(2):286–298, 2007.

[CGGR07] R. Chalodhorn, D. B. Grimes, K. Grochow, and R. P. N. Rao. Learning to walk through imitation. In M. Veloso, editor, *IJCAI*, pages 2084–2090, 2007.

[Cut89] M.R. Cutkosky. On Grasp Choice, Grasp Models and the Design of Hands for Manufacturing Tasks. *IEEE Trans. on Robotics and Automation*, 5(3), 1989.

[Dar72] C. Darwin. *On the Origin of Species by means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, 1872.

[DB98] A. D'Avella and E. Bizzi. Low dimensionality of supraspinally induced force fields. *Proceedings of the National Academy of Sciences of the United Stated of America*, 95(13):7711–7714, 1998.

[DFF+92] G. Di Pellegrino, L. Fadiga, L. Fogassi, V. Gallese, and G. Rizzolatti. Understanding motor events: a neurophysiological study. *Experimental Brain Research*, 91(1):176–180, 1992.

[DH97] P. Demartines and J. Herault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Trans. Neural Netw.*, 8(1):148–154, 1997.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DLB96] B. Douville, L. Levison, and N. Badler. Task-level Object Grasping for Simulated Agents. *Presence*, 5(4):416–430, 1996.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[DSRI06] S. Degallier, C. Santos, L. Righetti, and A. Ijspeert. Movement generation using dynamical systems: a humanoid robot performing a drumming task. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS06)*, 2006.

[dST04] V. de Silva and J.B. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Stanford University, USA, 2004.

[EMMT04] A. Egges, T. Molet, and N. Magnenat-Thalmann. Personalised real-time idle motion synthesis. In *Pacific Conference on Computer Graphics and Applications*, pages 121–130. IEEE Computer Society, 2004.

[ES03] G. ElKoura and K. Singh. Handrix: animating the human hand. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 110–119, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[FL95] C. Faloutsos and K.-I. Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Mchael Carey and Donovan Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174. ACM Press, 1995.

[FO71] K. Fukunaga and D.R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, 20(2):176–183, 1971.

[Fom09] A. T. Fomenko. *A Short Course in Differential Geometry and Topology*. Cambridge Scientific Pub., New York, 2009.

[FP03] A. C. Fang and N. S. Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):417–426, July 2003.

[FT87] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[FvT01] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 251–260, New York, NY, USA, 2001. ACM.

[GALP07] C. Goldfeder, P. K. Allen, C. Lackner, and R. Pelossof. Grasp planning via decomposition trees. In *ICRA*, pages 4679–4684. IEEE, 2007.

[GBT04] P. Glardon, R. Boulic, and D. Thalmann. Pca-based walking engine using motion capture data. In *CGI '04: Proceedings of the Computer Graphics International*, pages 292–298, Washington, DC, USA, 2004. IEEE Computer Society.

[GC91] A. Griewank and G. F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, Philadelphia, PA, 1991.

[GFFR96] V. Gallese, L. Fadiga, L. Fogassi, and G. Rizolatti. Action recognition in the premotor cortex. *Brain*, 119(2):593–609, 1996.

[GHBK09] S. Geidenstam, K. Huebner, D. Banksell, and D. Kragic. Learning of grasping strategies from box-based 3d object approximations. In *Proceedings of Robotics: Science and Systems*, 2009.

[GL93] F. Glover and M. Laguna. Tabu search. In C. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, Oxford, England, 1993. Blackwell Scientific Publishing.

[Gle98] M. Gleicher. Retargetting motion to new characters. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42, New York, NY, USA, 1998. ACM.

[GMHP04] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popovic. Style-based inverse kinematics. *ACM Trans. Graph.*, 23(3):522–531, 2004.

[Gra98] F. Sebastian Grassia. Practical parameterization of rotations using the exponential map. *journal of graphics tools*, 3(3):29–48, 1998.

[GUS99] H. Geyer, P. Ulbig, and S. Schulz. Use of evolutionary algorithms for the calculation of group contribution parameters in order to predict thermodynamic properties: Part 2: Encapsulated evolution strategies. *Computers and Chemical Engineering*, 23(7):955 – 973, 1999.

[H-A10] H-ANIM. Humanoid animation working group. www.hanim.org, last visited on May 1st 2010., 2010.

[HBAJ08] G. Heumer, H. Ben Amor, and B. Jung. Grasp recognition for uncalibrated data gloves: A machine learning approach. *Presence: Teleoperators & Virtual Environments*, 17(2):121–142, 2008.

[Her02] L. M. Herman. Vocal, social, and self-imitation by bottlenosed dolphins. In K. Dautenhahn and C. L. Nehaniv, editors, *Imitation in animals and artifacts*, pages 63–108. MIT Press, Cambridge, MA, USA, 2002.

[HGCB08] M. Hersch, F. Guenter, S. Calinon, and A. Billard. Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6):1463–1467, 2008.

[HHB07] D. Hein, M. Hild, and R. Berger. Evolution of biped walking using neural oscillators and physical simulation. In *RoboCup 2007: Proceedings of the International Symposium*, LNAI. Springer, 2007.

[HPP05] E. Hsu, K. Pulli, and J. Popović. Style translation for human motion. *ACM Trans. Graph.*, 24(3):1082–1089, 2005.

[HRE+08] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–11, New York, NY, USA, 2008. ACM.

[HS98] H. Heuer and J. Sangals. Task-dependent mixtures of coordinate systems in visuomotor transformations. *Experimental Brain Research*, 119(2), 1998.

[HUWK07] D. Holz, S. Ullrich, M. Wolter, and T. Kuhlen. Multi-contact grasp interaction for virtual environments. In *Proceedings 4. Workshop Virtuelle und Erweiterte Realität der GI-Fachgruppe VR/AR*, pages 101–112, 2007.

[HWBO95] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 71–78, New York, NY, USA, 1995. ACM.

[IAF06] L. Ikemoto, O. Arikan, and D. Forsyth. Knowing when to put your foot down. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 49–53, New York, NY, USA, 2006. ACM.

[IFoR10] International Federation of Robotics. IFR statistical department homepage. www.worldrobotics.org, last visited on May 1st 2010., 2010.

[IH03] C. Igel and M. Hüsken. Empirical evaluation of the improved rprop learning algorithms. *Neurocomputing*, 50:105–123, 2003.

[IMI08] S. Ikemoto, T. Minato, and H. Ishiguro. Analysis of physical human-robot interaction for motor learning with physical help. *Applied Bionics and Biomechanics (Special issue on Humanoid Robots)*, 5:213–223, 2008.

[INS02] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*, pages 1523–1530. MIT Press, 2002.

[IT04] M. Ito and J. Tani. On-line imitative interaction with a humanoid robot using a mirror neuron model. In *ICRA*, pages 1071–1076. IEEE, 2004.

[JAHV10] B. Jung, H. Ben Amor, G. Heumer, and A. Vitzthum. Action capture: A VR-based method for computer animation. In *Virtual Realities: Dagstuhl Seminar 2008*. Springer Verlag, 2010.

[JAHW06] B. Jung, H. Ben Amor, G. Heumer, and M. Weber. From motion capture to action capture: A review of imitation learning techniques and their application to VR-based character animation. In *Proceedings VRST 2006 - Thirteenth ACM Symposium on Virtual Reality Software and Technology*, pages 145–154, 2006.

[JM03] O. C. Jenkins and M. J. Matarić. Automated derivation of behavior vocabularies for autonomous humanoid motion. In *Autonomous Agents and Multiagent Systems (AAMAS 2003)*, pages 225–232, Melbourne, Australia, Jul 2003.

[KA08] M. Kass and J. Anderson. Animating oscillatory motion with overlap: wiggly splines. *ACM Transactions on Graphics*, 27(3), 2008.

[KAK+97] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for ai. *AI Magazine*, 18(1):73–85, 1997.

[KGP02] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 473–482, New York, NY, USA, 2002. ACM.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[KI95] S. B. Kang and K. Ikeuchi. A robot system that observes and replicates grasping tasks. In *ICCV '95: Proceedings of the Fifth International Conference on Computer Vision*, page 1093, Washington, DC, USA, 1995. IEEE Computer Society.

[KII94] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *IEEE. Trans. on Robotics and Automation*, 10(6):799–822, 1994.

[KL00] J. Kuffner and J. Latombe. Interactive Manipulation Planning for Animated Characters. In *Proceedings of Pacific Graphics*, 2000.

[KMI+80] N. Kamakura, M. Matsuo, H. Ishii, F. Mitsuboshi, and Y. Miura. Patterns of static prehension in normal hands. *The American journal of occupational therapy*, 34(7), 1980.

[KP06] P. G. Kry and D. K. Pai. Interaction capture and synthesis. *ACM Trans. Graph.*, 25(3):872–880, 2006.

[KR02] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 741–750, New York, NY, USA, 2002. ACM.

[KSG02] L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 97–104, New York, NY, USA, 2002. ACM.

[KSH01] T. Kohonen, M. R. Schroeder, and T. S. Huang. *Self-Organizing Maps*. Springer-Verlag New York, Inc., 2001.

[KT99] M. Kallmann and D. Thalmann. Direct 3D Interaction with Smart Objects. In *Proceedings ACM VRST 99, London*, 1999.

[KW78] J.B. Kruskal and M. Wish. *Multidimensional Scaling*. Sage Publications, Beverly Hills CA, 1978.

[Lat91] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[LBJK09] M. Lau, Z. Bar-Joseph, and J. Kuffner. Modeling spatial and temporal variation in motion data. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, pages 1–10, New York, NY, USA, 2009. ACM.

[LBZM06] H. Lipson, J. C. Bongard, V. Zykov, and E. Malone. Evolutionary robotics for legged machines: From simulation to physical reality. In T. Arai, R. Pfeifer, T. R. Balch, and H. Yokoi, editors, *IAS*, pages 11–18. IOS Press, 2006.

[LFP07] Y. Li, J. Fu, and N. Pollard. Data driven grasp synthesis using shape matching and task-based pruning. *IEEE Transactions on Visualization and Computer Graphics*, 13, 2007.

[LH99] J. Lee and B. V. Hout. Analyse de données non linéaires par résaux de neurones artificiels. mémoire d'ingénieur. Master's thesis, Université de Louvain-la-Neuve, Belgium, 1999.

[LHP05] C. K. Liu, A. Hertzmann, and Z. Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph.*, 24(3):1071–1081, 2005.

[LLDV00] J. A. Lee, A. Lendasse, N. Donckers, and M. Verleysen. A robust non-linear projection method. In *ESANN 2000, 8th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 26-28, 2000, Proceedings*, pages 13–20, 2000.

[LP83] T. Lozano-Perez. Robot programming. In *Proceedings of the IEEE*, volume 71, pages 821 – 841, 1983.

[LS99] J. Lee and S. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 39–48, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[LV07] J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer, New York; London, 2007.

[LWS02] Y. Li, T. Wang, and H. Shum. Motion texture: a two-level statistical model for character motion synthesis. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 465–472, New York, NY, USA, 2002. ACM.

[Mac67] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[Mah95] S. W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 1995.

[Mat02] M. J. Mataric. Sensory-motor primitives as a basis for imitation: linking perception to action and biology to robotics. In *Imitation in animals and artifacts*, pages 391–422. MIT Press, Cambridge, MA, USA, 2002.

[MC03] A. T. Miller and H. I. Christensen. Constraint stabilization for time-stepping approaches for rigid multibody dynamics with joints, contact and friction. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2262–2268, 2003.

[Mel96] A. N. Meltzoff. The human infant as imitative generalist: A 20-year progress report on infant imitation with implications for comparative psychology. In *Social Learning in Animals: The Roots of Culture*, pages 347–370, 1996.

[MF96] A. Moon and M. Farsi. Grasp quality measures in the control of dextrous robot hands. *Physical Modelling as a Basis for Control (Digest No: 1996/042), IEE Colloquium on*, pages 6/1–6/4, 1996.

[MI94] C. L. MacKenzie and T. Iberall. *The Grasping Hand*. Elsevier-North Holland, 1994.

[MIGB94] F. A. Mussa-Ivaldi, S F Giszter, and E Bizzi. Linear combinations of primitives in vertebrate motor control. *Proceedings of the National Academy of Sciences*, 91:7534–7538, 1994.

[MM77] A. N. Meltzoff and M. K. Moore. Imitation of facial and manual gestures by human neonates. *Science*, 198(4312):74–78, 1977.

[MM97] A. N. Meltzoff and M. K. Moore. Explaining facial imitation: A theoretical model. *Early Development and Parenting*, pages 179–192, 1997.

[MMMI07] D. Matsui, T. Minato, K. F. MacDorman, and H. Ishiguro. Generating natural motion in an android by mapping human motion. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems IROS 2007*, pages 609–616, 2007.

[MYN+07] T. Minato, Y. Yoshikawa, T. Noda, S. Ikemoto, H. Ishiguro, and M. Asada. Cb2: A child robot with biomimetic body for cognitive developmental robotics. In *Proceedings of the 2003 IEEE-RAS/RSJ International Conference on Humanoid Robots. Humanoids '07*, 2007.

[Nap56] J. Napier. The prehensile movements of the human hand. *The Journal of Bone and Joint Surgery*, 38b(4):902–913, 1956.

[OSOK09] K. Ogata, D. Shiramatsu, Y. Ohmura, and Y. Kuniyoshi. Analyzing the knack of human piggyback motion based on simultaneous measurement of tactile and movement data as a basis for humanoid control. In *IROS'09: Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, pages 2531–2536, Piscataway, NJ, USA, 2009. IEEE Press.

[Ozt02] E. Oztop. *Modeling the mirror: grasp learning and action recognition.* PhD thesis, University of Southern California, Los Angeles, CA, USA, 2002. Adviser-Arbib, Michael A.

[Par07] R. Parent. *Computer Animation, Second Edition: Algorithms and Techniques.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[PB02] K. Pullen and C. Bregler. Motion capture assisted animation: texturing and synthesis. *ACM Trans. Graph.*, 21(3):501–508, 2002.

[PG96] K. Perlin and A. Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 205–216, New York, NY, USA, 1996. ACM.

[Pol94] N. S. Pollard. *Parallel methods for synthesizing whole-hand grasps from generalized prototypes.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1994.

[PZ05] N. S. Pollard and V. B. Zordan. Physically based grasping control from example. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 311–318, New York, NY, USA, 2005. ACM.

[RB93] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 586–591. IEEE Press, 1993.

[RCB98] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl.*, 18(5):32–40, 1998.

[RFFG96] G. Rizolatti, L. Fadiga, L. Fogassi, and V. Gallese. Premotor cortex and the recognition of motor actions. *Cognitive Brain Research*, 3(2):131–41, 1996.

[RG91] H. Rijpkema and M. Girard. Computer animation of knowledge-based human grasping. *SIGGRAPH Comput. Graph.*, 25(4):339–348, 1991.

[RGBC96] C. Rose, B. Guenter, B. Bodenheimer, and M. F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 147–154, New York, NY, USA, 1996. ACM.

[RMGO96] J. O. Ramsay, K. G. Munhall, V. L. Gracco, and D. J. Ostry. Functional data analyses of lip motion. *Journal of the Acoustical Society of America*, 99:3718–3727, 1996.

[Röt07] F. Röthling. *Real Robot Hand Grasping using Simulation-Based Optimisation of Portable Strategies*. PhD thesis, Bielefeld University, Faculty of Technology, Bielefeld, Germany, May 2007.

[RPE+05] L. Ren, A. Patrick, A. Efros, J. K. Hodgins, and J. M. Rehg. A data-driven approach to quantifying natural human motion. *ACM Trans. Graph.*, 24(3):1090–1097, 2005.

[RS00] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

[RS06] J. Cornella R. Suarez, M. Roa. Grasp quality measures. Technical Report IOC-DT-P-2006-10, Universitat Politècnica de Catalunya, 3 2006.

[SAD97] H. Sadeghi, P. Allard, and M. Duhaime. Functional gait asymmetry in able-bodied subjects. *Human Movement Science*, 16(1):243–258, January 1997.

[Sam69] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 18(5):401–409, 1969.

[San00] T. D. Sanger. Human arm movements described by a low-dimensional superposition of principal component. *The Journal of Neuroscience*, 20:1066–1072, 2000.

[SCCH09] T. Shiratori, B. Coley, R. Cham, and J. K. Hodgins. Simulating balance recovery responses to trips based on biomechanical principles. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 37–46, New York, NY, USA, 2009. ACM.

[SCF06] A. Shapiro, Y. Cao, and P. Faloutsos. Style components. In *GI '06: Proceedings of Graphics Interface 2006*, pages 33–39, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[Sch19] G. Schlesinger. Der Mechanische Aufbau der Künstlichen Glieder. In M. Borchardt et al., editors, *Ersatzglieder und Arbeitshilfen für Kriegsbeschädigte und Unfallverletzte*, pages 321–661. Springer-Verlag: Berlin, Germany, 1919.

[Sch99] S. Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

[SF97] J. F. Soechting and M. Flanders. Flexibility and repeatability of finger movements during typing: Analysis of multiple degrees of freedom. *Journal of Computational Neuroscience*, 16(2-3):29–46, January 1997.

[SFS98] M. Santello, M. Flanders, and J. F. Soetching. Postural hand synergies for tool use. *The Journal of Neuroscience*, 18(23):10105–10115, 1998.

[SHP04] A. Safonova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph.*, 23(3):514–521, 2004.

[SOM04] A. Sud, M. A. Otaduy, and D. Manocha. Difi: Fast 3d distance field computation using graphics hardware. *Comput. Graph. Forum*, 23(3):557–566, 2004.

[ST94] R. M. Sanso and D. Thalmann. A Hand Control and Automatic Grasping System for Synthetic Actors. *Computer Graphics Forum*, 13(3):167–177, 1994.

[TBS04] C. Thurau, C. Bauckhage, and G. Sagerer. Synthesizing movements for computer game characters. In C. E. Rasmussen, H. H. Bülthoff, B. Schölkopf, and M. A. Giese, editors, *DAGM-Symposium*, volume 3175 of *Lecture Notes in Computer Science*, pages 179–186. Springer, 2004.

[TdL00] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319 – 2323, 2000.

[Tho98] E. L. Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *Psychological Review Monographs*, 8, 1898.

[Tho63] W. H. Thorpe. *Learning and instinct in Animals.* Methuen, London, 2nd edition edition, 1963.

[TIS04] J. Tani, M. Ito, and Y. Sugita. Self-organization of distributedly represented multiple behavior schemata in a mirror system: reviews of robot experiments using rnnpb. *Neural Netw.*, 17(8-9):1273–1289, 2004.

[TJ81] F. Thomas and O. Johnston. *The Illusion of Life: Disney Animation.* Abbeville Press, New York, 1981.

[TJD07] J. Triesch, H. Jasso, and G. O. Deák. Emergence of mirror neurons in a model of gaze following. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 15(2):149–165, 2007.

[TN03] K. Tatani and Y. Nakamura. Dimensionality reduction and reproduction with hierarchical nlpca neural networks-extracting common space of multiple humanoid motion patterns. In *ICRA*, pages 1927–1932. IEEE, 2003.

[TNNI08] J. Tani, R. Nishimoto, J. Namikawa, and M. Ito. Codevelopmental learning between human and humanoid robot using a dynamic neural-network model. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 38(1):43–59, 2008.

[TS55] C. L. Taylor and R. J. Schwarz. The Anatomy and Mechanics of the Human Hand. *Artificial Limbs*, 2:22–35, 1955.

[TYS+06] W. Takano, K. Yamane, T. Sugihara, K. Yamamoto, and Y. Nakamura. Primitive communication based on motion recognition and generation with hierarchical mimesis model. In *ICRA*, pages 3602–3609. IEEE, 2006.

[UAT95] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure animation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 91–96, New York, NY, USA, 1995. ACM.

[UKS89] Y. Uno, M. Kawato, and R. Suzuki. Formation and control of optimal trajectory in human multijoint arm movement. *Biological Cybernetics*, 61(2):89–101, June 1989.

[VAHJ09] A. Vitzthum, H. Ben Amor, G. Heumer, and B. Jung. Action description for animation of virtual characters. In *6. Workshop Virtuelle und Erweiterte Realität*. GI-Fachgruppe VR/AR, 2009.

[VBA+09] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '09)*, October 2009.

[WB97] A. Witkin and D. Baraff. Physically based modelling: Principles and practice. Course Notes of ACM Siggraph, 1997.

[WHAJ06] M. Weber, G. Heumer, H. Ben Amor, and B. Jung. An animation system for imitation of object grasping in virtual reality. In *Proceedings of Advances in Artificial Reality and Tele-Existence, 16th International Conference on Artificial Reality and Telexistence, ICAT*, pages 65–76. Springer, 2006.

[WK88] A. Witkin and M. Kass. Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 159–168, New York, NY, USA, 1988. ACM.

[WP95] A. Witkin and Z. Popovic. Motion warping. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 105–108, New York, NY, USA, 1995. ACM.

[WW01] S. Womble and S. Wermter. A mirror neuron system for syntax acquisition. In *ICANN '01: Proceedings of the International Conference on Artificial Neural Networks*, pages 1233–1238, London, UK, 2001. Springer-Verlag.

[XKZD09] Z. Xue, A. Kasper, J. M. Zoellner, and R. Dillman. An automatic grasp planning system for service robots. In *Proceedings of the 14th International Conference on Advanced Robotics*. IEEE, 2009.

[YLv07] K. Yin, K. Loken, and M. van de Panne. Simbicon: simple biped locomotion control. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 105, New York, NY, USA, 2007. ACM.

[Yur94] D. Yuret. From genetic algorithms to efficient optimization. Technical Report AITR-1569, MIT AI Laboratory, 6, 1994.

[ZH02] V. B. Zordan and J. K. Hodgins. Motion capture-driven simulations that hit and react. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 89–96, New York, NY, USA, 2002. ACM.

[ZRV04] J. C. Zagal, J. Ruiz-del-Solar, and P. Vallejos. Back-to-Reality: Crossing the reality gap in evolutionary robotics. In *IAV 2004: Proceedings 5th IFAC Symposium on Intelligent Autonomous Vehicles*. Elsevier Science Publishers B.V., 2004.

[ZSS96] T. R. Zentall, J. E. Sutton, and L. M. Sherburne. True imitative learning in pigeons. *Psychological Science*, 7(6):343–346, 1996.