Bootstrapping Movement Primitives from Complex Trajectories

Andre Lemme

Vorgelegt zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
Technische Fakultät, Universität Bielefeld
August 2014

Acknowledgment

Over the past five years I have received support and encouragement from a great number of individuals. Prof. Jochen J. Steil has been a mentor, colleague, and friend. His guidance has made this a thoughtful and rewarding journey.

I would like to thank our secretaries Anke Kloock, Heike Leifhelm and Ruth Moradbakhti for taking care of the administrative aspects of my PhD. Special thanks to Stefan Krüger, Michael Götting and Michael Johannfunke, who provided quick and professional technical support.

In addition, I like to thank Felix Reinhart, Andrea Soltoggio and Klaus Neumann for their support and many productive discussions about my research. I have learned much during our conversations. In my daily work I have been blessed with a friendly and cheerful group of fellow students. Especially, Christian Emmerich, Arne Nordmann and Stephan Rüther, who provided needed encouragement and with whom I shared many great memories.

Last but far from least I thank my family for their emotional support and especially my fiance and future wife Marina for being always by my side.

Thank you!

Printed on permanent paper according to DIN-ISO 9706.

Abstract – Bootstrapping Movement Primitives from Complex Trajectories

Andre Lemme

The impressive adaptive behavior of humans to approach tasks in everyday life is a desirable ability also for robotic applications. This behavior, especially the applied motion skills, is not available right after birth, but it is the result of a learning process. This learning process can acquire new motion skills or refine existing skills with growing experience (i.e., training). The goal of this thesis is to build a learning architecture which enables a robot to learn motion skills by means of a library of smaller motion building blocks. At first, the representation of movement shapes by *movement primitives* is discussed. It is shown that the movement primitives can be represented and learned with neural networks. In this context of movement primitives, many state-of-the-art movement learning methods are available. To compare the features and characteristics of these models, a novel benchmark framework is developed that allows systematic testing in standardized tasks according to human likeness and precision criterion. A decomposition algorithm is introduced to identify and extract similar primitives in complex trajectories. This algorithm uses a movement primitive library for the decomposition of complex trajectories. The *decomposition* is instrumental in the *creation* of new movement primitives and in the *refinement* of existing ones.

In a final step, these concepts are assembled in a versatile architecture that implements a higher level open-end learning process. It uses statistics from the decomposition algorithm such as what movement primitives are used and in what order they are needed to compose a similar complex trajectory. This information is exploited for the bootstrapping of new movement primitives from complex trajectories. The versatility and effectiveness of the architecture is evaluated in complex handwriting scenarios and also in a robotic application using the humanoid robot iCub.

CONTENTS

CHAPTER 1

INTRODUCTION

In every day life humans are able to control their bodies to perform complex motion tasks with phenomenal ease. These tasks can reach from a single handshake or waving to drawing, writing and complex object manipulation. These motion abilities are not present right after birth. Human infants are not able to move in a very coordinated way, which indicates a motion learning process that starts in the first years of their life and proceeds over years. One interesting example for the motion learning ability of young children is the process of learning how to write. The learning process starts in the first classes of school with simple tasks like reproducing single letters, before the complexity is increased and sequences of letters (i.e. words) are practiced. Many school books describe tasks which help the pupils to learn the art of cursive writing by tracing a shape of a letter e.g. [Merz et al., 2012]. These tasks provide information such as the direction of movement, together with indication of the start and end points. These instructions help to trigger an autonomous learning process in which the pupils can improve their motion skills by tracing letters multiple times. The way how humans are able to adapt their motion skills to complex tasks, as e.g. learning how to write, is most impressive and desirable to understand. The transfer of this ability to adapt to new complex tasks would be very beneficial in robotics and might open a path to multiple applications of 'learning' robots.

Since long ago, people have been fascinated by artificial beings that move and behave like humans. In the 15^{th} century, Leonardo da Vinci had plans for an artificial knight with limited motion capabilities like sitting up and functionality for arm movements. Of course, this mechanism did not move autonomously. Only with the help of a human pulling the strings, the robot (or puppet) was able to move (details can be found in [Moran, 2006]). Science fiction movies present different examples of humanoid robots or androids (e.g. Lieutenant Commander Data of the starship Enterprise [Luokkala, 2014]) which learn and move autonomously with impressive human like motion abilities that even surpass human beings. The development from an artificial knight to humanoid robots, maybe even similar to Commander Data, show an interesting theme, to which the work of this thesis is contributing to.

As a research field, humanoid robotics aims for robotic entities that serve as human companions and assist them in their daily life. This goal requires a repertoire of flexible motions and behavioral abilities which can cope with the complexity of the real world. Typically, robotic experts are in charge of programming robots to solve specific tasks in well defined environments [Argall et al., 2009]. This approach becomes very costly and does not scale, if redundant robotic systems are used with high degrees of freedom (DoF). It becomes more costly and time consuming, if the programming needs to be repeated every time the task conditions change, which also limits the adaptability to new unforeseen situations. Especially in human-robot interaction scenarios it is desirable to have robots which do not only move accurately along pre-programmed paths, but show motion behavior similar to humans in order to enhance the predictability and acceptance of the robot to the human interaction partner [Oztop et al., 2005; Chaminade et al., 2005].

Programming-by-demonstrations (PbD) is an alternative approach to prepare a robotic system for given tasks and increasingly important in the robotics community [Billard et al., 2008]. This learning

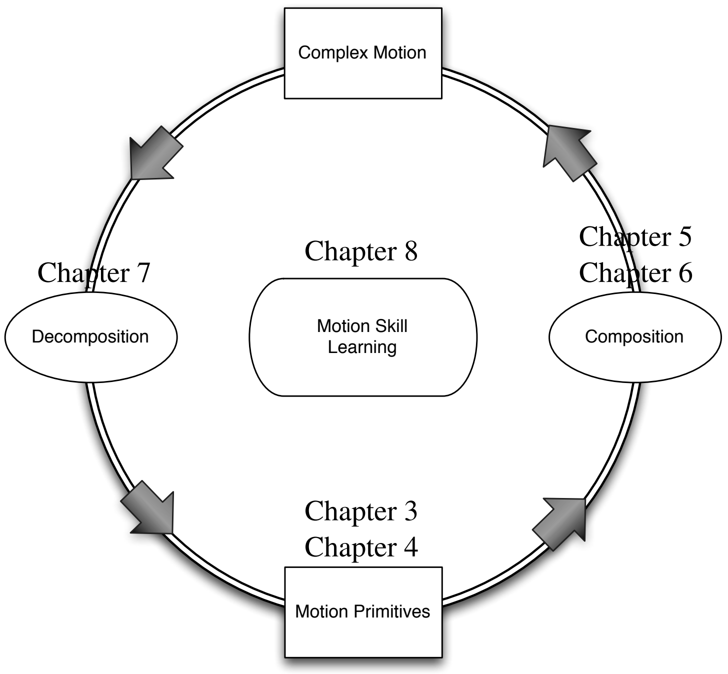


Fig. 1.1: Motion skill learning is a complex process discussed in this thesis. The endeavor starts in Chapter 3, 4, where motion primitives representations are in the research focus. How to compose these motion primitives to create complex motions are introduced in Chapter 5,6. To find the considered motion primitives again in the complex motion is described in Chapter 7. This processing cycle of composition and decomposition can be exploited for learning motion skills and is evaluated in Chapter 8.

paradigm allows to find important motion features, which are relevant for a task by solely analyzing recorded trajectory data and generalize them to new situations. In other words, the user of the robot demonstrates the task (see [Argall et al., 2009]) either with his or her own body, observed by a tracking system or in physical interaction with the robot itself i.e. kinesthetic teaching. It is most likely that human motion features are identified and mimicked by the robot from the recorded data, although the morphology of the robot may not be the same as the tutor's.

In context of learning motion skills, it is an established paradigm to use motion primitives (e.g. [Pastor et al., 2009; Mühlig et al., 2012; Reinhart et al., 2012]). One key idea is to find representations of motions which can be generalized towards different scenarios and allow to cope with changing situations. Such building blocks or motion primitives are expected to be used by humans to compose complex motions [Mussa-Ivaladi and E., 2000; Flash and Hochner, 2005]. Motion primitives on their own are limited and small units which need to be utilized in concert, such that more complex motions can be generated. For a new complex motion which is perceived, the necessary motion primitives to reproduce the complex motion are not directly clear. This means that the identification and decomposition of complex movements becomes necessary. The challenge is to build up a library of primitives, to sequence the available primitives according to a given task, and to use them to represent more complex motions, i.e. to perceive complex motions with known primitives.

The aim of this research is to develop a motion control architecture able to bootstrap a compact representation of motion primitives from complex trajectories and to generalize these to new tasks.

Thus, the overall objective of this thesis is to develop concepts and methods for autonomous learning of a primitive library, which create and refine motion skills from observations. An overview of the state-of-the-art approaches to represent motion primitives together with the biological interpretation is given in Chapter 2. In particular the standard learning paradigm and the general implementation of motion primitives in a learning architecture are described.

A schematic view for a general motion skill learning cycle developed in this thesis is given in Fig. 1.1. It illustrates a processing circle, where motion primitives are composed to complex motions, but also that complex motions are decomposed into motion primitives. By following this processing circle motion skill learning will be executed.

One specific example of a motion primitive representation is discussed in Chapter 3, where a neural learning approach is used. The work focuses on learning vector field in a neural network, such that it can be safely used as an autonomous dynamical system to represent motion primitives. The learning approach uses mechanisms to incorporate stability constraints into the learning process. Especially, in learning scenarios where only sparse trajectory data is provided, it becomes essential to incorporate stability constraints to provide stable dynamical systems for motion generation.

Due to the recent advent of a tremendous variety of new approaches for learning motion primitives, it became necessary to provide standardized tests for systematic comparisons. Therefore, a novel benchmark framework is proposed in Chapter 4, which can evaluate these approaches and can identify strengths and weaknesses of the different motion primitive representations. The delivered performance is evaluated accordingly to robotic features like precision but also analyses the human likeness of each approach.

After the discussion of motion primitive representations, the composition of motion primitives to complex motions moves into focus. Therefore, a primitive library approach which provides a possibility to store and to sequence primitives in a cheap and efficient fashion is proposed in Chapter 5.

As a next step towards the bootstrapping of movement primitives, concepts to create new and refine old primitives are introduced in Chapter 6. A non-autonomous dynamic system representation for motion primitives is used together with a neural network learning paradigm. It is shown that the combination of signal-depending noise and a global state representation is beneficial to produce a similar motion learning behavior as it is observed in human data.

An important component towards the targeted learning architecture is the decomposition of complex motions. In Chapter 7, an algorithm is introduced that uses a given set of motion primitives stored in motion primitive library and decomposes complex motions with a heuristic. This decomposition method provides information on how to compose the motion primitives to generate a similar complex motion.

Finally, the gathered techniques are applied to an autonomous learning cycle (as depicted in Fig. 1.1) in which motion primitives are bootstrapped from complex trajectories. In Chapter 8, it is shown that motion skills can be learned by bootstrapping a suitable motion primitive library. This learning cycle is then embedded into a larger motion skill learning architecture in the context of a robotic application with the humanoid robot iCub.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, the basic concepts of *motion primitives* are introduced from different perspectives. Thereby, the idea of complex motions being represented by building blocks denoted as motion primitives is discussed. Within the framework of motion primitives, possible categories of motion representation depending on the required command signals and the level of abstraction are defined. These categories are first discussed in relation to biological concepts such as motor synergies, motor primitives and movement primitives and are then illustrated by several mathematical models based on dynamical system theory.

The first question is: how many types of these motion primitive representations are in the human brain or mammal brain. This question is addressed in Sec. 2.1. The state-of-the-art methods to represent motion primitives in robotics and their learning approaches are discussed in Sec. 2.2. Possible architectural structures on how to use motion primitives for complex behavior generation are discussed in Sec. 2.3.

2.1 The idea of motion primitives from a biological perspective

The general idea of motion primitives is an established concept in the motor control community. Motion primitives describe small motion units that can be used as efficient building block to create complex motions. This section introduces important characteristics of these building blocks and discriminates between different types of motion primitives.

2.1.1 Observations of human arm motions and their invariant features

The analysis of human motions to identify principles of human motion generation is one of the major goals in the motor control community. In the literature, many invariant features of human motions have been discovered, e.g. [Wolpert et al., 1995]. These invariant features can be expressed by means of motion speed and geometric shape [Flash and Hogan, 1985; Flash, 1987]. One of the prominent features is the bell shaped form of the speed profile found in quasi straight line motions shown in [Morasso, 1981; Atkeson and Hollerbach, 1985]. In general, the common observations of human motion behavior is that the human brain seems to favor smooth motions, which is captured by, e.g. the minimum-jerk or minimum-variance model (see [Todorov, 2004] for a review).

Relations between motion features are captured in e.g. the isochrony principle [Viviani and Schneider, 1991] and Fitts' Law [Fitts, 1954]. The isochrony principle specifies that the average velocity of the motion is increased, if the size of the motion increases as well, i.e. the motion duration stays approximately the same. The Fitts' Law quantifies the relation between the movement amplitude and the target size, which means that accuracy constraints are taken into account. Isochrony principle together with Fitts' Law result in a prediction model for the target precision. Another regularity can be found in hand motions which tend to slow down, if the shape of the trajectory becomes curved. This tendency is quantified by the two-thirds power law [Lacquaniti et al., 1983], which predicts the hand's speed to be proportional to the curvature of the path of the motion trajectory raised to the power of minus one third.

The observation of such invariant motion features, indicates the organization of complex motions in smaller building blocks. These motion primitives capture the aforementioned features and define general principles for the generation of motions [Flash and Hochner, 2005]. As an example, a motion primitive can be defined solely with kinematic features in context of hand trajectories in Cartesian space, which can be also called e.g. sub-movements [Rohrer and Hogan, 2003]. One kinematic feature can be the peak of a speed profile, where each peak indicates the presence of one sub-movement. An alternative primitive definition is described by dynamics which consists of static force fields [Giszter and Kargo, 2001]. Although it is not entirely clear what drives the human motor system, it is known that human motions have these characteristics described by these features. In Chapter 4, these features are used as a baseline for the comparison of computational motion generation methods in terms of human likeness.

2.1.2 Distribution of motion primitives in the motor hierarchy

In this section, the focus is on hypotheses where the motion primitive representations may reside and what kind of functionality they provide in the human motor hierarchy.

In the wide variety of motions, two categories can be identified. First, discrete motions and second rhythmic motions. Discrete motions generate trajectories, which connect a start and end point e.g. reaching to an object. Rhythmic or oscillatory motions describe "closed" trajectories like circles or ellipses. Oscillatory motions are often used in locomotion to model gait behavior of bipedal or quadruple leg robots [Ijspeert, 2008]. But also in drawing or scribbling motions are examples for oscillatory motions. In [Degallier and Ijspeert, 2010], a detailed review is given comparing different categories of models for motion generation combining discrete and rhythmic primitives. Primitives on a neuronal level corresponds to a neuron assembly of spinal or cortical neurons. These can correspond to central pattern generators (CPGs reviewed in [Ijspeert, 2008]) or to point-to-point motions (reaching motions). These variations of motion primitives, describe either a cyclic or point attractor [Schaal et al., 2003a]. Motions like walking or drumming are easy to categorize, however, it is also possible that reaching motions are executed in a periodic way, which makes it more difficult to classify them. Waving motions for example can be described as a single rhythmic motion, but also as two reaching motions sequenced one after the other. In experiments with functional magnetic resonance imaging (fMRI) evidence was found which encourages the believe that different representations of motions are justified. These experiments showed separated cortical activity if rhythmic or discrete motions are performed [Schaal et al., 2004].

Primitives can describe motions on multiple levels in the motor hierarchy. For instance, the paradigm of *motor synergies* located at the muscular level can be found in the literature [d'Avella et al., 2003a; d'Avella and Bizzi, 2005; Latash et al., 2007; Bizzi et al., 2008]. The sheer amount of muscles in the human body allows humans to move in a very flexible way that enables complex motions to be optimized for each task that needs to be solved. Motor synergies describe the timely coordinated activation of muscles depending on the task. By using this synergy representation the need to generate detailed time courses for all elemental variables in a given task is reduced. Therefore, this representation may reduce the computational costs for the central nervous system (CNS) [d'Avella and Lacquaniti, 2013].

In the higher level of the motor hierarchy, concepts like *motor primitives* and *movement primitives* are described in the literature. Motor primitives can be seen as building blocks, which can be composed to generate the entire motion repertoire [Wolpert and Kawato, 1998] together with discrete muscle synergies. These concepts are used, but do not consistently describes the same behavior. One definition of motor primitives [Mussa-Ivaldi et al., 1994] is that trajectories are learned on joint level, whereas motion primitives describe trajectories learned in hand (end-effector) coordinate system.

In the field of neuroscience, motion skills of increasing complexity shall be achieved by blending and sequencing of motion primitives by utilizing a continuous stream of activation signals [Luksch et al., 2012c]. These activation signals should then indicate the sequence of motion primitives which are used either in parallel or in a serial blending. Representation of sequences and the selection of those might be done in the CNS [Rhodes et al., 2004]. In this thesis, the concepts for discrete point-to-point reaching

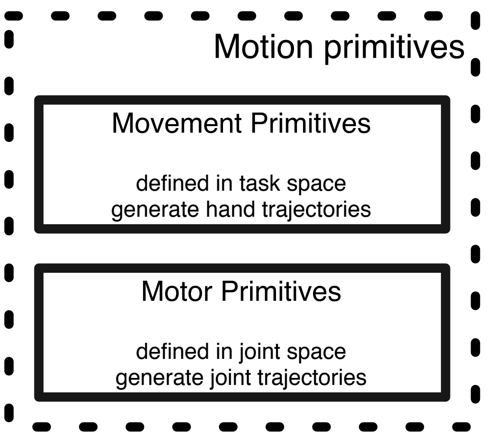


Fig. 2.1: In this thesis *motion primitives* denote two types of primitives. The first type is the *movement primitive* which represents motions in a task space for hand trajectories. The second type is the *motor primitive* and represents motions on the joint level and generates joint trajectories.

motions are in the focus. Fig. 2.1 illustrates the relation the terminology of motion primitives, movement primitives and motor primitives as they are used in this thesis.

2.2 Motion primitive representation and learning in robotics

In this section the robotic applications of motion primitives are introduced and discussed according to: (i) What representations of motion primitives can be used? (ii) What features must be incorporated in the primitive representation? And (iii) how can motion primitives be adapted to a given motion shape in a learning process.

2.2.1 Motion primitive representation in dynamical systems

In the context of robotic systems, the representation of reaching movements is typically given in form of dynamical systems. Two conceptually different dynamical systems are considered given by first order differential equations:

(A) The non-autonomous dynamical systems:

$$\dot{\mathbf{u}} = g(\alpha, \mathbf{u}, t) \tag{2.1}$$

are considered, where $\mathbf{u}$ is a state vector, t is the time parameter and α gives a modulation parameter, which can be used to modulate the speed of the system.

(B) The autonomous dynamical system:

$$\dot{\mathbf{u}} = g(\alpha, \mathbf{u}), \tag{2.2}$$

where no explicit time dependency exists.

Nonlinear dynamical systems appear to be one of the most promising candidates as computational basis for exploitation of flexible motion capabilities featured by modern humanoid robots [Mühlig et al., 2012; Pistillo et al., 2011; Billard et al., 2008]. Point-to-point reaching motions modeled by dynamical systems can provide a library of basic motion primitives, which can be successfully applied to generate motions in a variety of manipulation tasks [Moro et al., 2012; Ude et al., 2010].

An example, which uses a second order nonlinear dynamical system, is the Vector Integration To Endpoint model (VITE). It was originally developed by [Bullock and Grossberg, 1988, 1989] and is designed to simulate biological plausible straight motions using a muscular system. However, there is no learning methodology provided, which can adapt to non straight line motions. In the following, means to approximate complex shapes by machine learning methods are considered.

2.2.2 Learning how to move from experts

Manual programming of robot motions often requires a large amount of engineering knowledge about both the task and the robot. This issue becomes particularly non-intuitive when dealing with dexterous humanoid robots with many DoFs [Biggs and MacDonald, 2003]. In recent years, human-like motions in, not only humanoid robots, but also industrial robots became desirable. The advantage of robots with human motion capabilities is that humans can predict easier where the robot is going. This becomes especially important if robots start to work in close proximity with the human [Oztop et al., 2005; Chaminade et al., 2005].

Therefore, it is widely recognized that exploiting human knowledge about motion skills either by reproduction of recorded human movements [Ude et al., 2004; Riley et al., 2003], by training through teleoperation [Babic et al., 2011] or by observation and subsequent imitation [Calinon et al., 2007; Yamashita and Tani, 2008; Pastor et al., 2009; Kulic et al., 2011] is a key to progress towards rich motion skills in robotics [Schaal and Sternad, 1998; Schaal et al., 2003a]. Typically, primitives are learned from a small number of trajectories demonstrated by a human teacher, which is referred to as e.g. learning from demonstrations [Atkeson and Schaal, 1997; Argall et al., 2009], imitation learning [Schaal, 1999] or programming by demonstrations [Billard et al., 2008]. These learning approaches aim at the generation of an invariant internal representation which can generalize the observed motion to modified or new tasks.

One issue is that the quality of the learning results strongly correlates with the quality of the shown demonstrations. To counter act this issue, prior knowledge of the task is used in the learning approach where "wrong" demonstrations are corrected or ignored. In human-robot interaction scenarios, the robot needs to perform the motion even if the interaction partner is touching the robot or pushing him per accident without injuring the interaction partner or breaking itself. Therefore, the most important prior knowledge in learning from demonstrations is that the learned dynamics are stable with regard to perturbations and does not result in unpredictable behavior of the robot. Due to the large variety of different methods an overview of them is discussed which represent three different ways on how to approach these requirements in imitation learning.

2.2.3 Computational models for representing and learning movement primitives

In the last decades, many computational models for representing motion primitives have been proposed, e.g. [Ijspeert et al., 2003; Giese et al., 2009; Mukovskiy et al., 2013; Moro et al., 2011; Grave and Behnke, 2012; Paraschos et al., 2013; Khansari-Zadeh and Billard, 2011, 2014; Calinon et al., 2013; Reinhart, 2012]. This thesis focuses on movement primitives represented by dynamical systems. Nonlinear dynamical systems are a promising basis to represent motion primitives. Besides their ability to display diverse behavior, it is of major importance that such systems work in a stable manner. To find a suitable dynamical system that ensures these requirements simultaneously is particularly hard. Two prominent learning approaches are described, (i) spring damper systems and (ii) vector field based approaches.

Dynamic Movement Primitives

A widely known approach is called Dynamic Movement Primitives (DMP) [Ijspeert et al., 2002, 2003; Schaal, 2006; Ijspeert et al., 2013], which is a technique to generate motions with non-autonomous dynamical systems that models a spring-damper system. This analytically well understood dynamical system with good stability properties is considered, where the dynamics can be adapted by a nonlinear force term to achieve a desired attractor behavior. This approach provides a very accurate nonlinear estimate of a given trajectory which is robust to perturbations while ensuring global stability.

The original DMP approach uses a transformation system (following the notation of [Ijspeert et al., 2013]):

$$\tau\ddot{\mathbf{u}} = \alpha_u(\beta_u(\mathbf{g}-\mathbf{u})-\dot{\mathbf{u}}) + \underbrace{\mathbf{f}(s)(\mathbf{g}-\mathbf{u}_0)s}_{\text{perturbation}}\ , \tag{2.3}$$

coupled with a canonical system:

$$\tau\dot{s} = -\alpha_s s, \tag{2.4}$$

where α_u, β_u are stiffness and damping constants $\beta_u = \alpha_u/4$ the system is critically damped and converges towards the goal $\mathbf{g}$. The stability of this dynamical system is ensured in case that the perturbation $\mathbf{f}$ becomes zero at the end of the movement, which results in a linear convergence to the goal point. The smooth switch from nonlinear to linear dynamics is controlled by a phase variable s. The phase variable can be seen as external stabilizer which in return distorts the temporal pattern of the dynamics.

A variation of the DMP transformation system is given by [Park et al., 2008].

$$\tau\ddot{\mathbf{u}} = K(\mathbf{g}-\mathbf{u}) - D\dot{\mathbf{u}} - Ks(\mathbf{g}-\mathbf{u}_0) + \underbrace{K\mathbf{f}(s)}_{\text{perturbation}}\ , \tag{2.5}$$

where K, D are stiffness and damping constants with $D = 2\sqrt{K}$ to generate a critically damped system. It is also coupled with a canonical system as described above in Eq. (2.4). The major difference is that the transformation system in Eq. (2.5) is invariant to the relative position of the start and goal position. In [Park et al., 2008], the properties of this transformation system are described in more detail.

The transformation system Eq. (2.5) is a linear dynamical system which is perturbed by a non linearity function f to represent arbitrarily shaped and smooth reaching motions. Typically, the function f can be approximated by a mixture of Gaussian basis functions. The state representation R for the Gaussian basis function networks (GBFs) is given by:

$$f_i(s) = \frac{\sum_{j=1}^{R}\Phi_j(s)\mathbf{W}_{ij}^{out}}{\sum_{j=1}^{R}\Phi_j(s)} = (\mathbf{W}_i^{out}\mathbf{h}(s)), \tag{2.6}$$

with $\Phi_i(s) = exp(-l_i(s-c_i)^2)$ representing the nonlinearity in the DMP approach. c_i and l_i are fixed a priory. Due to linear dependence of f on $\mathbf{W}^{out} \in \mathbb{R}^{K\times R}$ it simplifies the use of learning methods and allows to use any linear regression technique. One exemplary learning approach is the Linear Weighted Regression (LWR) [Schaal and Atkeson, 1998] to solve this regression problem for motion primitives.

Stable Estimator of Dynamical Systems

An exemplary approach that uses an autonomous dynamical system with a vector field representation is the Stable Estimator of Dynamical System (SEDS) [Khansari-Zadeh and Billard, 2011]. The SEDS approach uses a Gaussian Mixture Models (GMM) representation which encodes vector fields where global asymptotic stability is ensured.

The dynamical system is defined by a nonlinear combination of linear dynamical systems. The open parameters $\Theta = \{\pi, \mu, \Sigma\}$ are the prior π, the mean μ and the covariance matrix Σ, where the mean μ and the covariance Σ are defined by:

$$\mu = \begin{bmatrix}\mu_u \\ \mu_{\dot{u}}\end{bmatrix} \ \ \&\ \ \Sigma = \begin{bmatrix}\Sigma_u & \Sigma_{u\dot{u}} \\ \Sigma_{\dot{u}u} & \Sigma_{\dot{u}}\end{bmatrix}.$$

Then each linear dynamical system is given by:

$$z(\mathbf{u}) = (\mu_{\dot{\mathbf{u}}} + \Sigma_{\dot{\mathbf{u}}\mathbf{u}}\Sigma_{\mathbf{u}}^{-1}(\mathbf{u}-\mu_u)). \tag{2.7}$$

These linear dynamical systems are combined by:

$$\hat{\mathbf{u}} = \hat{f}(\mathbf{u}) = \sum_{m=1}^{M} g^m(\mathbf{u})z^m(\mathbf{u}), \tag{2.8}$$

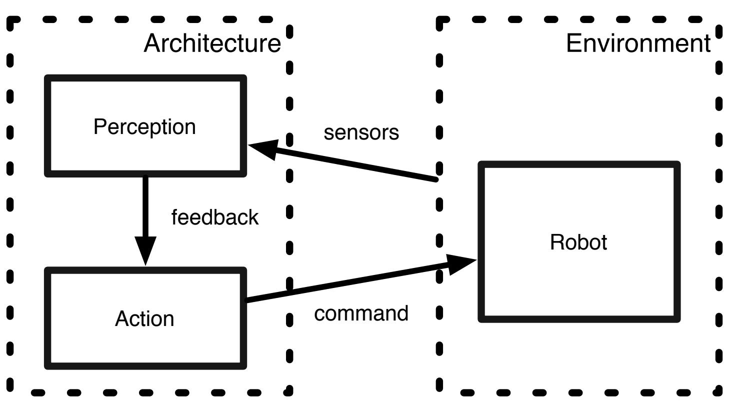


Fig. 2.2: A general motion architecture has to organize the integration of perception and action into one coherent framework. Actions can be formulated in form of motion primitives, which can generate commands for the robot. If the environment has uncertainties perceiving the environment is crucial. The feedback can then be used to alter the planed actions which move the robot.

where M is the number of Gaussian functions. The nonlinear weighting term $g(\mathbf{u})$, where $0 < g(\mathbf{u}) \leq 1$ models the contribution of each linear dynamical system in Eq. (2.7) to the combined system. The learning method can minimize either the mean square error:

$$\arg\min_{\Theta} E(\Theta) = \frac{1}{2N} \sum_{n=1}^{N} ||\hat{\dot{\mathbf{u}}}^n - \dot{\mathbf{u}}^n||^2, \tag{2.9}$$

or the log-likelihood to build a model as described in [Khansari-Zadeh and Billard, 2011]. These error functions are solved under stability constraints derived from a quadratic Lyapunov function. This optimization problem can be formulated as a *nonlinear* programming problem and can be solved with iterative optimization techniques.

2.3 Architectural perspective on motion skill learning

Humans and animals are capable of learning, perfecting and reproducing complex trajectories that allow them to perform a variety of tasks, from coordinated body movements to catching, and particularly in humans, object manipulation, writing and drawing. The mechanisms underlying motion skills, from the learning of basic motion primitives to their organization in higher-level cognitive structures, are fundamental in understanding how humans accomplish advanced motion skills [Schack, 2004].

Motion primitives are only limited building blocks which can create complex trajectories if they are used in combination, e.g. by sequencing motion primitives one after the other or by executing them in parallel which is also denoted by *blending*. The question is how such building blocks can be organized in a motion skill architecture?

2.3.1 General architecture structures and primitive-based control

In general, a motion skill architecture needs to combine perception and actions in order to control a robot in an environment where uncertainties exists as illustrated in Fig. 2.2, which is also biologically motivated [Fuster, 2004]. The perception needs to process sensory inputs to provide necessary feedback for the actions. In this case actions are represented as motion primitives which generate commands to move the robot in the environment.

Typically, motion primitives are used in a hierarchical design, where either motor or movement primitives build the basic control units interfacing the joint control directly or the task space control. In the latter case, an inverse kinematics solver is needed which can map the task space trajectories to joint angle trajectories.

The HAMMER (hierarchical attentive multiple models for execution and recognition) architecture motivates the organization of motion in a hierarchical structure [Demiris and Khadhouri, 2006] and it aspires to be biologically plausible. It is hiding details of motion generation in the lower level of motion primitives. These are encoded as pairs of forward and inverse models. The forward model produces prediction of the current state whereas the inverse models produces a motor command to move the current state to the target/goal. Adding new primitives becomes possible through a self-exploration mechanism inspired by motor babbling in infants or through imitating others. In [Demiris and Khadhouri, 2006], forward and inverse models are represented by Bayesian belief networks (BBN). An extension of the HAMMER model utilizes a reformulated minimum variance model [Harris and Wolpert, 1998], such that it can be used in this architecture called HAMMER-MV architecture [Demiris and Simmons, 2006] which allows a more biological motion generation. Another biological inspired approach was earlier introduced in [Billard and Matarić, 2001]. This approach models abstractions of brain regions involving also visuo-motor control in a hierarchy of neural networks each representing e.g. the spinal cord, motor cortex and the cerebellum.

In [Ito and Tani, 2004; Tani et al., 2004; Paine and Tani, 2004], a recurrent neural network with parametric biases (RNNPB) is presented. This network can generate and recognize behavior patterns and is inspired by the neural mirror system. A highly nonlinear dynamical system models a self-organized mapping between parameter vectors and different behavior patterns in a single neural network. It is shown that multiple cyclic motion patterns can be represented as limit cycle in the internal memory of the neural network. Point-to-point motion with smooth blending behavior is also possible. The continuous time recurrent neural network (CTRNN) is proposed in [Yamashita and Tani, 2008]. The recurrent neural network is structured, where each part is running on a different temporal scale without spatial hierarchical structure.

Primitive based control is introduced in [Speeter, 1991], where grasping motions are implemented using the abstraction of primitives. In [Inamura et al., 2001, 2003], a learning architecture is introduced in which whole body motion capture data is used to extract movement primitives and sequences of these primitives, to compose complex whole body motions. The segmentation of complex motions is done by identifying high acceleration points (using second order derivati). These motion segments are then classified by a probabilistic neural network with radial basis functions (RBFs) to generate movement primitive symbols. These symbols are then used to model the seen motion by hidden Markov models (HMMs). A complex motion is reproduced by using the identified set of trajectories and sequencing them again by simply attaching the segments one to another. To achieve a smooth trajectory which can be reproduced by the robot, a dynamic filter [Inamura et al., 2003] is applied that fulfills dynamic constraints. In most previous work on learning motion primitives, the observed data is directly given or transformed into a suitable task space [Calinon et al., 2007; Asfour et al., 2008; Pastor et al., 2009; Kulic et al., 2011], in most cases tool tip or hand coordinates or even more task specific coordinates [Mühlig et al., 2009, 2012]. In this task space, a compact representation of the movement is learned where invariance can be achieved by representing movements in relative task coordinates, e.g. by placing the goal of a reaching movement at the origin of the coordinate system. For motion generation, this task space representation is then *modulated*, for instance transformed according to the visual recognition of an object to be manipulated. The movement is deployed on the robot via inverse kinematic solvers. This scheme provides high flexibility, in particular for free goal-directed movements, and efficient methods have been provided to resolve the kinematic redundancies [Chiaverini, 1997; Liégeois, 1977; Gienger et al., 2005, 2008]. In [Luksch et al., 2012a], a control approach for motion skills is introduced. These motion skills are modeled by continuously combining motion primitives, where the motion primitives can be designed for different reference spaces. These primitives are represented by dynamical systems and coordinated by a continuous stream of control signals. In [Reinhart et al., 2012], it is discovered that generalization strongly scales with variations of the motion skill architecture, i.e. how the different motion representations are combined. The paper argues in a qualitative evaluation of different architectural configurations of representational layers for a particular scheme with a late spatial modulation of skills in favor of optimal generalization from few demonstrations.

Motion skills which use several motion primitives in this context can be associated with sensory input during the motion skill execution. These machine learning methods are denoted as *associative skill memories* [Pastor et al., 2013]. Depending on the sensory information the associative skill memory can be used differently. However, the main application is to repeat the motion skill as stereotypical as possible. In Chapter 5, a similar concept is used to store movement primitives and the execution sequence in one shared representation of a neural network approach.

2.3.2 Identifying primitives in complex trajectories

In the neurocognitive literature, the concept of motor resonance motivates that human infants use their own motion representation to perceive and represent observed motion behavior (see [Berthier, 1996] and [Paulus, 2014] for a review). So the question is how can a skill learning architecture process complex trajectories into smaller motion primitive representation in robotics? There are three ways on how to approach this issue denoted as *decomposition*. First, the decomposition can be designed to segment a complex trajectory according to predefined features or cost functions. Second, the decomposition is designed according to the motion primitive methods:

The detection of critical points counts to the first design scheme of decomposition approaches and is used in [Mohan et al., 2011]. Critical points can be used to build a symbolic representation of the complex trajectory, where a set of critical points can then again be transformed into a complex trajectory. In [Endres et al., 2013], the motion power law describing the relation of the velocity and the curvature (see Sec. 2.1.1) is used for unsupervised segmentation of endpoint trajectories. Velocity and curvature are used as parametrization of the seen trajectory and is exploited for the unsupervised segmentation. Segments can be identified by using a Gaussian observation model denoted as Bayesian binning (BB). Also Hidden Markov Models are used in [Kohlmorgen and Lemm, 2001], where segmentation is conducted by following the changes of the probability density in a sliding window. By using predefined features for the decomposition it becomes necessary to represent those also in the motion primitives to make sure that similar features are generated. As a consequence, motion primitives do not exceed a certain complexity which is limited by the predefined features.

Some approaches, e.g. [Mann et al., 2002; Hellbach et al., 2009; Kohlmorgen and Lemm, 2001], focus on the demonstrated trajectory that is analyzed and decomposed, employing polynomial decomposition [Mann et al., 2002] according to a specific cost function. One intrinsic feature and limitation of motion primitives is that they generate basic rather than arbitrary long and convoluted trajectories. As a consequence, recent research has focused on the problem of combining simple primitives to form more complex and longer trajectories [Mann et al., 2002; Hellbach et al., 2009; Konidaris et al., 2010; Peters and Schaal, 2008; Konidaris et al., 2012]. Particularly those based on reinforcement learning, assume a given cost function to a specify the task and learn the motion primitive representation accordingly as shown in [Peters and Schaal, 2008; Konidaris et al., 2010, 2012]. Algorithms that combine primitive or shape-identification, trajectory segmentation and on-line learning have also be proposed e.g. in [Kulic et al., 2011; Mohan et al., 2011; Konidaris et al., 2012] to integrate various sub-problems in more capable learning algorithms.

In the recent years, decomposition approaches are introduced which use a set of motion primitives to represent complex motions. In [Meier et al., 2011, 2012], DMPs are used to represent motion primitives. A likelyhood is calculated at each timestep for all motion primitives, which then specifies the best match for the current segment of the complex trajectory. Alternatively, in [Grave and Behnke, 2012], a similar approach is introduced where Hidden Markov Models are applied for the decomposition and representation of the motion primitive.

Chapter 7 contributes to this topic by presenting an algorithm which uses a set of motion primitives to represent complex trajectories. The decomposition only focuses on geometrical properties of trajectories, while it is agnostic to the velocity profiles. This apparent limitation in reality allows for a more flexible interpretation of trajectories, which may not be necessarily determined by the velocity profile used during generation.

CHAPTER 3

MOVEMENT PRIMITIVES IMPRINTED IN NEURAL NETWORKS

In Chapter 2, the central role played by movement primitive in human and robot motion generation is emphasized. In this chapter, a new autonomous dynamical systems approach is presented, where an underlying neurally imprinted vector field is learned from sparse data to drive the dynamical system. In general the data-driven approximation of vector fields that encode dynamical systems is a difficult task in machine learning. If data is sparse and given in form of velocities derived from few trajectories only, then there are state-space regions where no information on the vector field and its induced dynamics is available. Generalization towards such regions is meaningful only if strong biases are introduced, for instance assumptions on global stability properties of the to-be-learned dynamics.

In this proposed approach, asymptotic stability of the induced dynamics is explicitly enforced through utilizing prior knowledge from Lyapunov's stability theory, in a predefined workspace. The learning of vector fields is constrained through point-wise conditions, derived from a suitable *Lyapunov function candidate*, which is first adjusted towards the training data. The approach is analyzed in a scenario where trajectories from human handwriting motions are learned and generalized. In addition, robust motion generation is demonstrated after learning from robotic data obtained from kinesthetic teaching of the humanoid robot iCub. Finally, potential downsides and advantages of this new learning approach are discussed. The results and conceptional ideas are also published in [Lemme et al., 2013; Neumann et al., 2013a; Lemme et al., 2014a].

3.1 Learning vector field representations from sparse data

From a general machine learning point of view, the approximation of vector fields from sparse data to represent dynamical systems, e.g. encoding of quantitative flow visualization [Gharib et al., 2002], optical flow in computer vision [Verri and Poggio, 1989] or force fields in motor control [Mussa-Ivaldi and Giszter, 1992; Giszter and Kargo, 2001], is an important but also persistently hard task for learning algorithms.

In recent work, vector fields were applied to learn and generate complex motions for robots [Heinzmann and Zelinsky, 2003; Corteville et al., 2007]. In such scenarios, training data typically consist of only few trajectories and thus leave many regions in the state space with no information of the desired vector field. Generalization towards regions subject to sparse sampling is challenging, because small errors in the approximation of the vector field can get amplified during integration and can lead to the divergence of the dynamical system, which is an unwanted and unpredictable behavior.

Thus, a strong model bias is needed for generalization which has to be derived from prior knowledge about the underlying dynamics. In [Mussa-Ivaldi, 1992], a superposition of irrotational basis fields is used to approximate a variety of vector patterns, where it is assumed that the data originate from the

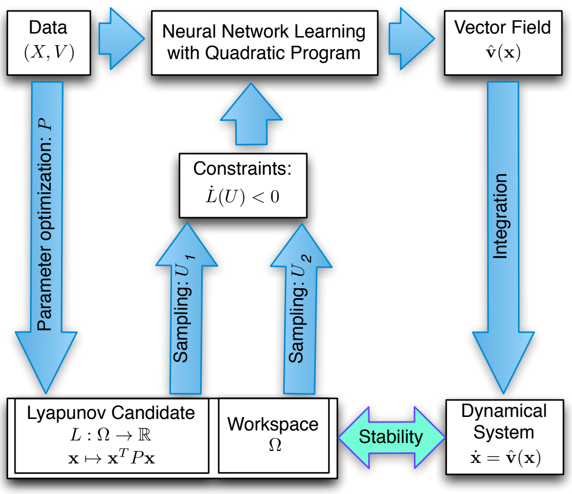

Data
(X, V)
Neural Network Learning with Quadratic Program
Vector Field
$\hat{\mathbf{v}}(\mathbf{x})$
Parameter optimization: P
Constraints:
$\dot{L}(U) < 0$
Integration
Sampling: U_1
Sampling: U_2
Lyapunov Candidate
$L : \Omega \to \mathbb{R}$
$\mathbf{x} \mapsto \mathbf{x}^T P \mathbf{x}$
Workspace
Ω
Stability
Dynamical System
$\dot{\mathbf{x}} = \hat{\mathbf{v}}(\mathbf{x})$

Fig. 3.1: Schematic view of the proposed approach to learn vector fields with respect to constraints derived from prior knowledge. The learning is separated into three main steps: i) predefine a proper Lyapunov candidate through parameter optimization and ii) use this function to sample inequality constraints that are implemented by a quadratic program learning the data. iii) Add constraints to restrict the motion to stay in the defined workspace. The resulting dynamical system approximates the data and is asymptotically stable in the defined workspace after learning.

gradient of a potential function. Kuroe and Kawakami introduced a combination of neural networks to reconstruct vector fields where prior knowledge of inherent vector field properties is used to enhance the accuracy [Kuroe and Kawakami, 2007, 2009].

In context of motion generation from vector fields, one prominent approach is the stable estimator of dynamical systems (SEDS) which is described in Sec. 2.2.3. This learning approach represents vector fields by a Gaussian mixture of linear dynamical systems. Learning is achieved by solving a nonlinear constrained optimization problem formulated as a quadratic program. The learned dynamical system then complies to a specific quadratic Lyapunov function. The main advantage of this method is that the learned dynamics are provably globally asymptotically stable. On the downside, the stability constraints may be too restrictive with respect to the motion that shall be learned. If the training data and the stability constraints contradict, accurate learning of the desired motion is prevented. However, the learning of dynamics that satisfy desired Lyapunov functions and guarantees stability without interfering with the data is so far only solved for special cases and remains difficult in case of using a dynamical systems represented by vector fields.

These issues are partly addressed in [Reinhart et al., 2012]. Here a neural network approach is used to learn from demonstrations and to generate motions for the humanoid robot iCub. The accuracy performance and the stability are addressed by two separately trained but superimposed neural networks. The first network approximates the data while the second network addresses stability by learning a velocity field, which implements a contraction towards the desired movement trajectory. However, the superposition of two networks seems complex for representing only one motion. Additionally, no guarantee for stable motion generation is given.

The contribution of this chapter is to introduce of stability theory of dynamical systems, described by Lyapunov, in neural networks learning in context of stable motion generation. It is based on the idea to represent time-independent vector fields in one neural network that lead to asymptotically stable dynamics in a predefined workspace. This approach is schematically illustrated in Fig. 3.1. The learning is separated into three steps: First, construct a suitable stability function (i.e. a so called Lyapunov function) through parameter optimization towards the data. Second, use the constructed Lyapunov function to obtain inequalities constraints for learning. Third, add inequality constraints which ensure that the dynamics cannot leave a predefined region. The inequality constraints are implemented by a quadratic program, which minimizes the error between the training data and the output of the network. To keep the number of used constraints to a minimum, a sampling algorithm identifies problematic regions and adds constraints until the dynamical system is stabilized. Thus, the resulting vector field induces stable dynamics by construction. The flexibility of this new learning approach is demonstrated in a rigorous analysis that evaluates the own learning properties and also compares the learned dynamical systems to the SEDS approach. Additionally, it is demonstrated that the new approach generates smooth and accurate motions in a kinesthetic teaching scenario with the humanoid robot iCub.

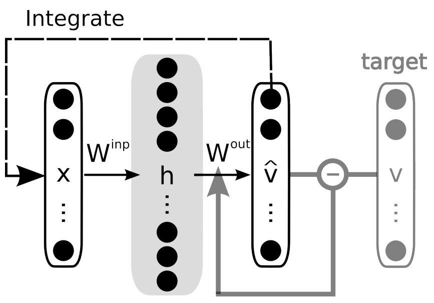


Fig. 3.2: ELM with its three layer structure used in an integration loop. Only the read-out weights are trained.

3.2 Neurally imprinted vector fields for motion generation

In this section, sparse data in form of trajectories are considered. In the following, the task and the learning setup are introduced.

3.2.1 Dynamical system based motions

In this work, point-to-point motions are considered which can be described as trajectory data given by the state variable $\mathbf{x}(t)$ with time t. The trajectories are considered to be driven by an underlying autonomous dynamical system, which can be represented by vector fields:

$$\dot{\mathbf{x}} = \mathbf{v}(\mathbf{x}) \quad , \mathbf{x} \in \Omega \quad , \tag{3.1}$$

where the state variable $\mathbf{x}(t) \in \Omega \subset \mathbb{R}^d$ with dimension d and a workspace Ω. Because point-to-point motions are considered, it is assumed that the vector field $\mathbf{v}(\mathbf{x})$ is nonlinear and continuous with a single asymptotically stable point attractor $\mathbf{x}^*$ with $\mathbf{v}(\mathbf{x}^*) = 0$ in Ω. The limit of each trajectory in Ω thus satisfies:

$$\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}^* : \forall \mathbf{x}(0) \in \Omega \quad . \tag{3.2}$$

The key question of this work is how to learn $\mathbf{v}$ as a function of $\mathbf{x}$ by using demonstrations for training and ensure its asymptotic stability at target $\mathbf{x}^*$ in Ω. The estimate is denoted by $\hat{\mathbf{v}}$ in the following. The generation of motions can then be computed by numerical integration of $\dot{\mathbf{x}} = \hat{\mathbf{v}}(\mathbf{x})$, where $\mathbf{x}(0) \in \Omega$ denotes the starting point of the motion:

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \Delta t \cdot \mathbf{v}(\mathbf{x}(t)), \tag{3.3}$$

where Δt is a time constant for discretization of the continuous dynamics.

3.2.2 Vector field representation and learning with extreme learning machines

For the estimation of $\hat{\mathbf{v}}$ the neural architecture depicted in Fig. 3.2 is described. The figure shows a single hidden layer feed-forward neural network, where $\mathbf{x} \in \mathbb{R}^d$ denotes the input, $\mathbf{h} \in \mathbb{R}^R$ the hidden, and $\hat{\mathbf{v}} \in \mathbb{R}^d$ the output neurons. The input is connected to the hidden layer by the input matrix $\mathbf{W}^{inp} \in \mathbb{R}^{R\times d}$. The read-out matrix is given by $\mathbf{W}^{out} \in \mathbb{R}^{d\times R}$. For input $\mathbf{x}$, the output $\hat{\mathbf{v}}$ is thus given by:

$$\hat{\mathbf{v}} = \mathbf{W}^{out}\mathbf{h}(\mathbf{x}) = \mathbf{W}^{out}\sigma(\mathbf{W}^{inp}\mathbf{x}),$$

where $\sigma(a_i) = (1 + exp(-a_i - b_i))^{-1}$ are sigmoid activation functions applied to the neural activities $a_i = \sum_{j=1}^{d} (\mathbf{W}^{inp})_{ij} x_j$ for all hidden neurons $i = 1,\dots,R$. The components of the input matrix and the biases are drawn from a random distribution and remain fixed after initialization. The following experiments reveal that this network architecture is particularly well suited for incorporation of stability constraints in the learning. The input weights $\mathbf{W}^{inp}$ and biases b_i are randomly initialized and remain fixed. This approach is called extreme learning machine (ELM) [Huang and Siew, 2006]. ELMs combine a nonlinear and high-dimensional random projection of inputs $\mathbf{x} \in \mathbb{R}^d$ into a hidden layer $\mathbf{h} \in \mathbb{R}^{R \gg d}$ with efficient linear regression learning of a perceptron-like read-out layer $\mathbf{W}^{out}$. Let

$D = (\mathbf{x}(k), \mathbf{v}(k)) : k = 1 \dots N_{\mathrm{tr}}$ be the dataset for training where N_{tr} is the number of samples in the dataset. Supervised learning for ELMs is restricted to the read-out weights $\mathbf{W}^{out}$. They are trained by minimizing the quadratic error with weight regularization:

$$\mathbf{W}^{out} = \arg\min_{\mathbf{W}}(\|\mathbf{W} \cdot \mathbf{H}(\mathbf{X}) - \mathbf{v}\|^2 + \varepsilon\|\mathbf{W}\|^2) \tag{3.4}$$

which is solved by:

$$\mathbf{W}^{out} = \left(\mathbf{H}\mathbf{H}^T + \varepsilon\mathbb{1}\right)^{-1}\mathbf{H}\mathbf{v}^T \ , \tag{3.5}$$

where $\mathbf{H} = (\mathbf{h}(\mathbf{x}(1)), \dots, \mathbf{h}(\mathbf{x}(N_{\mathrm{tr}}))) : k = 1 \dots N_{\mathrm{tr}}$ is a matrix harvesting the hidden states for each input $\mathbf{x}(k)$ in the training dataset, $\mathbb{1}$ is the identity matrix, and $\varepsilon > 0$ is a regularization parameter.

In Fig. 3.3, two examples of the learned estimation of a nonlinear dynamical system using the ELM are shown. Due to the generalization ability of the ELM both examples show that the motion can either converges to other spurious attractors away from the desired attractor (i.e. the target) or completely diverge from it. This observed performance is highly undesired because one of the most important feature in the case of point-to-point movements is to converge to a given target. This issue is addressed by the additional learning schema described in the following.

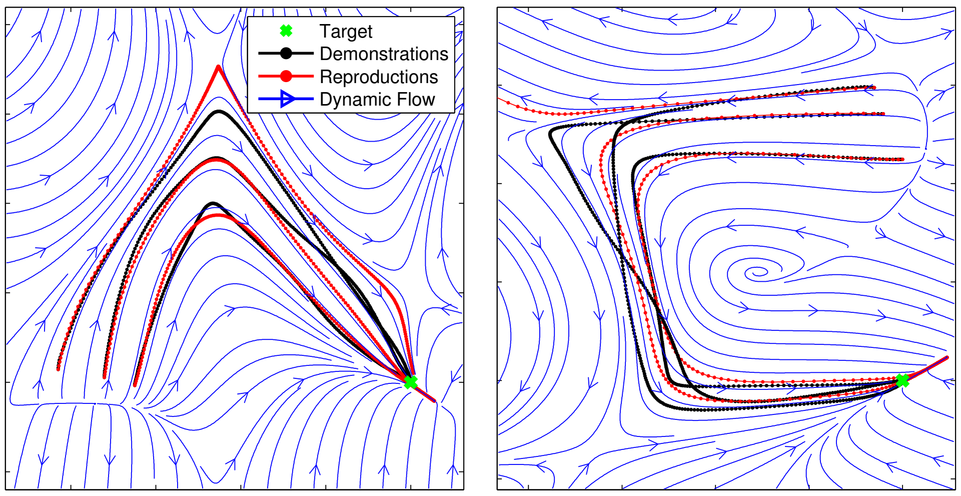


Fig. 3.3: Unstable estimation of dynamical systems through demonstrations. Data were taken from the LASA dataset: A-shape (left) and sharp-C (right). Note that the reproduced trajectories either diverge or converge to spurious attractors.

3.3 Implementation of asymptotic stability

Learning a vector field from a few training trajectories gives only sparse information of the shape of the entire vector field. Therefore, there is considerable need for generalization to spatial regions where no training data reside. Besides the generalization ability, particular feature need to be learned as well, e.g. the target should be described as a fixed-point attractor in the vector field. Learning this attractor without prior knowledge is especially hard, because the training data comprises of only a few training samples that encode the target.

3.3.1 Extreme learning machines with Lyapunov theory

In order to stabilize the dynamical system induced by the network, asymptotic stability conditions of arbitrary dynamical systems defined by Lyapunov are recalled: a dynamical system is asymptotically stable at fixed-point $\mathbf{x}^* \in \Omega$ in the compact region $\Omega \subset \mathbb{R}^d$ if there exists a continuous function $L : \Omega \to \mathbb{R}$

$$(\mathbf{i})\ L(\mathbf{x}^*) = 0 \qquad (\mathbf{ii})\ L(\mathbf{x}) > 0 : \forall \mathbf{x} \in \Omega, \mathbf{x} \neq \mathbf{x}^* \tag{3.6}$$

$$(\mathbf{iii})\ \dot{L}(\mathbf{x}^*) = 0 \qquad (\mathbf{iv})\ \dot{L}(\mathbf{x}) < 0 : \forall \mathbf{x} \in \Omega, \mathbf{x} \neq \mathbf{x}^* \ . \tag{3.7}$$

For now it is assumed that the function L satisfies condition (**i**)-(**iii**) and is called *Lyapunov candidate*. In order to obtain a learning algorithm for $\mathbf{W}^{out}$ that also respects the condition (**iv**) of the Lyapunov candidate L, The time derivative of L is taken to analyze this condition:

$$\dot{L}(\mathbf{x}) = \frac{\mathrm{d}}{\mathrm{d}t}L(\mathbf{x}) = (\nabla_{\mathbf{x}} L(\mathbf{x}))^T \cdot \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = (\nabla_{\mathbf{x}} L(\mathbf{x}))^T \cdot \hat{\mathbf{v}} \tag{3.8}$$

$$= \sum_{i=1}^{I} (\nabla_{\mathbf{x}} L(\mathbf{x}))_i \cdot \sum_{k=1}^{R} W_{ij}^{\mathbf{out}} \cdot f(a_j \sum_{k=1}^{I} W_{jk}^{\mathrm{inp}} x_k + b_j) < 0 \ . \tag{3.9}$$

Note that $\dot{L}$ is linear in the output parameters $\mathbf{W}^{out}$ irrespective of the form of the Lyapunov candidate L. For a given point $\mathbf{u} \in \Omega$, Eq. (3.8) and Eq. (3.9) define a linear constraint $\dot{L}(\mathbf{u}) < 0$ on the read-out parameters $\mathbf{W}^{out}$, which is implemented by a quadratic programming scheme introduced for ELMs in [Neumann et al., 2013b]. It is shown in [Neumann et al., 2013b] that a well-chosen sampling of points $U = (\mathbf{u}(1), \ldots, \mathbf{u}(N_{\mathrm{u}})) : \mathbf{u} \in \mathbb{R}^d$, is sufficient to generalize the incorporated discrete constraints to continuous regions in a reliable way.

The read-out weights $\mathbf{W}^{out}$ are trained by minimizing the quadratic error with weight regularization:

$$\mathbf{W}^{out} = \arg\min_{\mathbf{W}}(\|\mathbf{W} \cdot \mathbf{H}(\mathbf{X}) - \mathbf{v}\|^2 + \varepsilon \|\mathbf{W}\|^2) \tag{3.10}$$

$$\text{with subject to: } \dot{L}(U) < 0 \ , \tag{3.11}$$

where the matrix $\mathbf{H}(\mathbf{X}) = (\mathbf{h}(\mathbf{x}(1)), \ldots, \mathbf{h}(\mathbf{x}(N_{\mathrm{tr}})))$ collects the hidden layer states obtained from a given dataset $D = (\mathbf{X}, \mathbf{V}) = (\mathbf{x}(k), \mathbf{v}(k)) : k = 1 \ldots N_{\mathrm{tr}}$ for inputs $\mathbf{X}$ and the corresponding output vectors $\mathbf{V}$.

3.3.2 Positive invariant regions

The previous section introduced the implementation of inequalities, which are derived from a Lyapunov candidate L, at discrete points. These points are element of a predefined region Ω. However, stability according to condition (**iv**) is not stringently achieved in the entire space Ω without consideration of L. Even if the condition (**iv**) is valid in Ω, the state of the dynamical system $\dot{\mathbf{x}} = \hat{\mathbf{v}}(\mathbf{x})$ can possibly cross the border of Ω during numerical integration. In this case, stability cannot be guaranteed, since those parts of the input space outside Ω are not considered in the constraint implementation process. Stability is, in principle, enforced in the largest level set region of the candidate function L that is completely intersecting with the sampling region Ω. Each initial point outside this region is potentially subject to divergence. As mentioned before, the identification of such regions is difficult for arbitrary Lyapunov candidates.

Therefore an additional constraint is introduced. It is defined on the surface of Ω (denoted by $\partial\Omega$) which forces the learned dynamics to stay in the predefined region Ω. This region can principally have arbitrary shapes, however, only hypercubes are used where each point $\mathbf{x} \in \Omega$ on the surface of this cube $\partial\Omega$ is mapped onto an uniquely defined normal vector $\mathbf{n}(\mathbf{x}) \in \mathbb{R}^d$ with $\|\mathbf{n}(\mathbf{x})\| = 1$ that point outward of the cube. The resulting constraints are expressed as a scalar product between the normal vector $\mathbf{n}(\mathbf{x})$ and the network's output $\hat{\mathbf{v}}(\mathbf{x})$:

$$L_{\partial}(\mathbf{x}) := \mathbf{n}(\mathbf{x})^T \cdot \hat{\mathbf{v}}(\mathbf{x}) \leq 0 : \mathbf{x} \in \Omega \ . \tag{3.12}$$

Note that the scalar product has the same form as Eq. (3.8) and is thus linear in $\mathbf{W}^{out}$. It forces the network's dynamics to stay inside the hypercube implementing a positive invariant region (see Fig. 3.4).

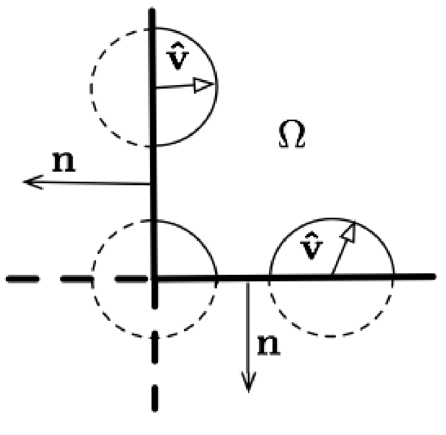


Fig. 3.4: Possible solutions, which can be chosen by the learning algorithm. The solid black lines give the left bottom corner of the rectangle illustrating the workspace border $\partial\Omega$. The center of each circle represent the corresponding position $\mathbf{x}$. Possible directions of $\hat{\mathbf{v}}$ are indicated by the solid part of each circle. Note that the corners are allowing only a small range for a valid solution of the dynamics.

3.3.3 Sampling strategy to implement efficiently the Lyapunov candidate

The following sampling strategy is introduced in order to minimize the number of samples needed for generalization of the local constraints towards the continuous region Ω. The dataset D for training and the region Ω where the constraints are supposed to be implemented are assumed to be given. As a first step ($k = 0$), the network is initialized and trained without any constraints (i.e. the sample matrices $U_i^k = U_i^0 = \emptyset$ are empty). In this case learning can be accomplished by linear regression, which is the standard learning scheme for ELMs, see Eq. (3.5).

In the next step, N_C samples $\hat{U} = \{\hat{\mathbf{u}}^1, \hat{\mathbf{u}}^2, \ldots, \hat{\mathbf{u}}^{N_C}\}$ are randomly drawn from a uniform distribution in Ω and $\partial\Omega$, respectively. Afterwards, the number of samples v_1 fulfilling (**iv**) of Lyapunov's conditions of asymptotic stability (see Eq. (3.8)) and the number of samples v_2 satisfying Eq. (3.12) are determined.

The sampling algorithm stops if more or equal than p percent of these samples fulfill the continuous constraints, i.e. $v_i/N_C \geq p$. Otherwise, the most violating sample $\hat{\mathbf{u}}$ for each constraint is added to the respective sample pool $U_i^{k+1} = U_i^k \cup \hat{\mathbf{u}}$ to constitute the new sample set U_i^{k+1}. The most violating sample $\hat{\mathbf{u}}$ thus maximizes either $\dot{L}(\hat{\mathbf{u}})$ or $L_\partial(\hat{\mathbf{u}})$ with respect to the Lyapunov candidate L. The obtained set of samples is then used for training. A pseudo code of the learning procedure is provided in Algorithm 3.1.

Algorithm 3.1 Sampling Strategy

```
Require: dataset $D$, region $\Omega$, counter $k = 0$, sample pools $U_i^k = \emptyset$, and ELM $\hat{\mathbf{v}}$ trained with $D$
1: repeat
2:     draw $N_C$ samples $\hat{U} = \{\hat{\mathbf{u}}^1, \hat{\mathbf{u}}^2, \ldots, \hat{\mathbf{u}}^{N_C}\}$
3:     $v_1 =$ no. of samples in $\hat{U}$ fulfilling condition (iv)
4:     $v_2 =$ no. of samples in $\hat{U}$ fulfilling Eq. (3.12))
5:     if $p > \frac{v_1}{N_C}$ then $U_1^{k+1} = U_1^k \cup \arg\max_{\mathbf{u} \in \hat{U}} \dot{L}(\mathbf{u})$
6:     if $p > \frac{v_2}{N_C}$ then $U_2^{k+1} = U_2^k \cup \arg\max_{\mathbf{u} \in \hat{U}} L_\partial(\mathbf{u})$
7:     train ELM with $D$ and $U_i^{k+1}$
8:     $k \leftarrow k + 1$
9: until $p \leq \frac{v_i}{N_C} : \forall i$
```

Note that the constraints at points U and the input samples $\mathbf{X}$ of the training data are not the same. For learning according to Eq. (3.10), it remains to select a set of samples U in addition to the training inputs $\mathbf{X}$. As mentioned before it was already shown in [Neumann et al., 2013b] that a well-chosen sampling of points U is sufficient for generalization of the incorporated discrete constraints towards a continuous region Ω and that stability can be proven ex-post after learning. This is possible since the neural network method allows analytical differentiation and exploits the linear dependency on the learning parameters in order to use a worst case approximation by means of Taylor polynomials. The approximation of the constraint surface is, however, computationally costly and should only be applied if required. It is also important to note that, in absence of the verification processes, stability is only given with a certain probability. Sampling and ex-post verification is only possible in finite regions and can only enforce local asymptotic stability.

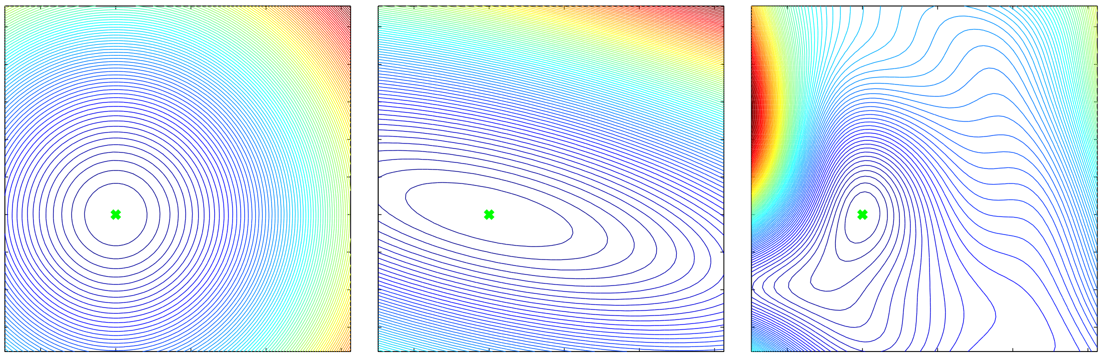

(a) $L_q = x^2$ (b) $L_P = x^T P x$ (c) $L = ?$

Fig. 3.5: Level sets of three different Lyapunov candidates, but which one to apply for learning?

3.4 Design of data-driven Lyapunov candidates

As a first example, a well known quadratic Lyapunov candidate is used, given by $L_q = (\mathbf{x}-\mathbf{x}^*)^T \mathbb{1}(\mathbf{x}-\mathbf{x}^*)$, where $\mathbb{1}$ denotes the identity matrix for stability implementation. This function is data independent, quadratic and fulfills conditions (**i**)-(**iii**) in Sec. 3.3.1. The Lyapunov candidate function L_q is only able to capture a limited class of dynamics, but more complex functions could be possible as depicted in Fig. 3.5. This raises the question: which Lyapunov candidate function is suitable for learning dynamical systems from complex but sparse data? In the following approaches are presented, which can be shaped accordingly to training data.

3.4.1 Quadratic Lyapunov candidates

As a first attempt to find a Lyapunov candidate with more complexity, the function L_q is parameterized and is considered in the following form:

$$L_P(\mathbf{x}) = \frac{1}{2}(\mathbf{x}-\mathbf{x}^*)^T P(\mathbf{x}-\mathbf{x}^*) \quad . \tag{3.13}$$

Note that (**i**)-(**iii**) are fulfilled if P is positive definite and symmetric. It is defined as:

$$P = \arg\min_{P\in\mathscr{P}} M(L_P) \quad . \tag{3.14}$$

M is defined according to the following sum over the training data:

$$M(L_P) = \frac{1}{N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \Theta\left[(\mathbf{x}^i(k))^T P \mathbf{v}^i(k)\right] \quad , \tag{3.15}$$

where the desired attractor $\mathbf{x}^* = 0$ and $\Theta : \mathbb{R} \to \{0,1\}$ denotes the heavy side function.

$$\Theta(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{otherwise.} \end{cases}$$

The minimization operator in Eq. (3.14) can be formulated as a nonlinear program, which allows to be solved by successive quadratic programming based on quasi Newton methods for optimization

[Bazaraaa et al., 2006]. The possible matrices are restricted to be an element of $\mathscr{P} := \{P \in \mathbb{R}^{d\times d} : P^T = P,\ \lambda_i \in [\alpha, 1],\ \lambda_i \text{ is EV of } G\}$[1], where $\alpha = 0.1$ is a small and positive scalar.

Note that for some shapes it is impossible to find an estimate that approximates the data accurately while simultaneously fulfilling the quadratic constraint given by the Lyapunov candidate L_q. Part of the training samples $(\mathbf{x}(k), \mathbf{v}(k))$ are violated by L_q, where violation means that the angle between the negative gradient of the Lyapunov candidate at point $\mathbf{x}$ and the velocity of the data sample $\mathbf{v}$ is bigger than 90°:

$$\sphericalangle(-\nabla L(\mathbf{x}(k)), \mathbf{v}(k)) > 90^\circ \Leftrightarrow (\nabla L(\mathbf{x}(k)))^T \mathbf{v}(k) > 0 \tag{3.16}$$

Note that the derivation in Eq. (3.9) shows that this is in contradiction to condition (**iv**) of Lyapunov's theorem.

In order to evaluate if a given Lyapunov candidate L is compliant with the training data D, the measure M in Eq. (3.15) is generalized to arbitrary Lyapunov candidates. The violation of the training data is defined as:

$$M(L) = \frac{1}{N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \Theta\left[(\nabla L(\mathbf{x}^i(k)))^T \cdot \mathbf{v}^i(k) \right] \quad , \tag{3.17}$$

only those samples $(\mathbf{x}^i(k), \mathbf{v}^i(k))$ are counted in M where the scalar product between $\nabla L(\mathbf{x}^i(k))$ and $\mathbf{v}^i(k)$ is positive and thus violating Lyapunov's condition (**iv**).

3.4.2 Learning Lyapunov candidates with neural networks

The application of standard candidates, i.e. quadratic Lyapunov candidates as in Fig. 3.5(a) or matrix parametrized candidates visualized in Fig. 3.5(b) are not satisfactory if the tasks demand an appropriate complexity. In theory, much more complex functions are possible and also desired, see Fig. 3.5(c).

To approach the issue of designing such a flexible Lyapunov candidate an ELM is considered which defines a scalar function $L_{\text{ELM}} : \mathbb{R}^d \to \mathbb{R}$. The major goal is to minimize the violation of the training data measured by Eq. (3.17) by making the negative gradient of this function follow the training data. The definition of a quadratic program suited for the new constraints is given by:

$$\frac{1}{N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \left(\| -\nabla L_{\text{ELM}}(\mathbf{x}^i(k)) - \mathbf{v}^i(k) \|^2 + \ldots \ + \varepsilon_{\text{RR}} \| W^{\mathbf{out}} \|^2 \right) \to \min_{W^{\mathbf{out}}} \quad , \tag{3.18}$$

subject to the following equality and inequality constraints corresponding to Lyapunov's conditions (**i**)-(**iv**) such that L_{ELM} becomes a valid Lyapunov candidate:

$$\begin{array}{ll} (\mathbf{a}) \quad L_{\text{ELM}}(\mathbf{x}^*) = 0 & (\mathbf{b}) \ L_{\text{ELM}}(\mathbf{x}) > 0 : \mathbf{x} \neq \mathbf{x}^* \\ (\mathbf{c}) \ \nabla L_{\text{ELM}}(\mathbf{x}^*) = 0 & (\mathbf{d}) \ \dot{L}_{\text{ELM}}(\mathbf{x}) < 0 : \mathbf{x} \neq \mathbf{x}^* \end{array} \tag{3.19}$$

where the time derivative in (**d**) is defined w.r.t the stable system $\dot{\mathbf{x}} = -\mathbf{x}$. The constraints (**b**) and (**c**) define inequality constraints which are implemented by sampling these constraints.

The gradient of the scalar function defined by the ELM is linear in $\mathbf{W}^{out}$ and given by:

$$(\nabla L_{\text{ELM}}(\mathbf{x}))_i = \sum_{j=1}^{R} \mathbf{W}_j^{out} f'\left(\sum_{k=1}^{d} \mathbf{W}_{jk}^{inp} x_k + b_j \right) \cdot (\mathbf{W}_{ji}^{inp}) \quad , \tag{3.20}$$

where f' denotes the first derivative of the Fermi function. A sampling strategy is defined which is very similar to the one defined in Sec. 3.3.3 (see Algorithm 3.1). This strategy is described in Algorithm 3.2.

The final dynamical system is acquired in two steps. First use one ELM to learn a suitable Lyapunov candidate (L_{ELM}) by using Algorithm 3.2. Then use a second ELM to implement the desired dynamical system by using the derived stability constraint from the first ELM representing the Lyapunov candidate.

[1] EV is used as abbreviation of eigenvalue.

Algorithm 3.2 Lyapunov Function Learning

```
Require: dataset $D$, region $\mathscr{S}$, counter $k = 0$, sample pools $U_1^k = \emptyset$ and $U_2^k = \emptyset$
Require: $L_{\text{ELM}} : \mathbb{R}^d \to \mathbb{R}$ trained with $D$ w.r.t. (a) and (c)
  repeat
    draw samples $\hat{U} = \{\hat{\mathbf{u}}^1, \hat{\mathbf{u}}^2, \ldots, \hat{\mathbf{u}}^{N_\text{C}}\}$
    $v_1$ = no. of samples in $\hat{U}$ fulfilling (2)
    $v_2$ = no. of samples in $\hat{U}$ fulfilling (4)
    if $p > \frac{v_1}{N_\text{C}}$ then $U_1^{k+1} = U_1^k \cup \arg\max_{\mathbf{u}\in\hat{U}} L_{\text{ELM}}(\mathbf{u})$
    if $p > \frac{v_2}{N_\text{C}}$ then $U_2^{k+1} = U_2^k \cup \arg\min_{\mathbf{u}\in\hat{U}} \dot{L}_{\text{ELM}}(\mathbf{u})$
    train ELM with $D$, $U_1^{k+1}$ and $U_2^{k+1}$ w.r.t. (a) and (c)
    $k \leftarrow k+1$
  until $p > \frac{v_1}{N_\text{C}}$ and $p > \frac{v_2}{N_\text{C}}$
```

3.5 Impact of stability-constraints on the resulting dynamical systems

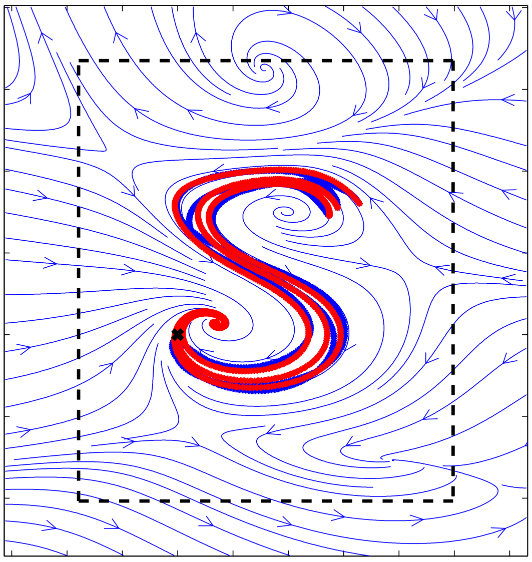
In order to analyze the impact of the learning with constraints, experiments are conducted, where the networks learn from human-demonstrated handwriting motions collected for motion primitive learning [Khansari-Zadeh, 2012a][2]. At first a single motion is considered, which is composed of three S-like trajectories with 250 samples each and the end-point located at the origin (see Fig. 3.6).

The ELM is initialized with $R = 100$ neurons in the hidden layer. The biases b_i and components of $\mathbf{W}^{inp}$ are initialized randomly drawn from the uniform distribution in $[-1, 1]$. The regularization parameter is set to $\varepsilon = 10^{-5}$. The parameterized Lyapunov candidate function is adapted to the training data according to Eq. (3.14). The set of constraints U consists of samples drawn from the region $\Omega = [-0.5, 2.1] \times [-1.1, 2.5]$, which covers the relevant region of the task space. Fig. 3.6(a) illustrates an example of learning the S-like trajectories by an ELM without the usage of explicit stability constraints. In the areas close to the demonstrations, the trajectories converge to an attractor next to the target. In other regions of the space, they either converge to spurious attractors or diverge. In contrast, Fig. 3.6(b) shows the same network setup but trained with the stabilization method using Eq. (3.13) with metric P chosen according to Eq. (3.14). Note that in the left upper corner of the local workspace, shown in Fig. 3.6(b), the dynamics can leave the controlled workspace. In Fig. 3.6(c) constraints are added to restrict the dynamics to the local region workspace Ω. The generated trajectories converge to the target, because the learning process enforces asymptotic stability at target $\mathbf{x}^*$. This ensures that the target is reached, if the movement is started from any point in the workspace Ω. The adopted Lyapunov candidate enables the accurate modeling of the dynamics. In the following a systematic evaluation of the new stability mechanism is conducted using two performance measures.

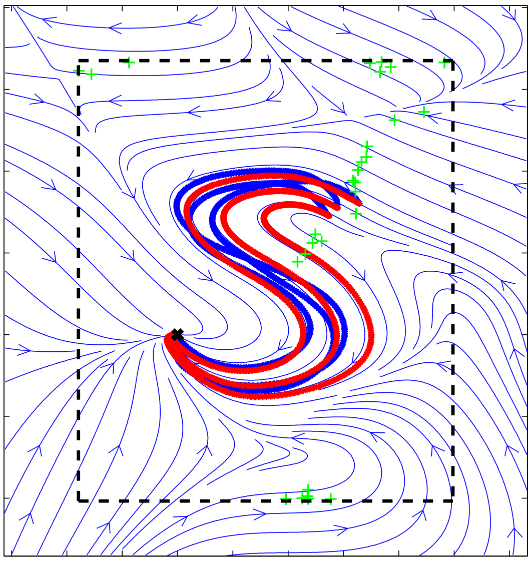
The first measure is the root mean square error $E_{\text{tr}} = \sqrt{\frac{1}{N_{\text{tr}}} \sum_k \|\mathbf{v}(k) - \hat{\mathbf{v}}(\mathbf{x}(k))\|^2}$ evaluated on the training data, which quantifies the ability to approximate the training data. The second and third measure quantify the stability of the dynamical system numerically. For these measures, the motion generation is simulated by choosing $N_\text{s} = 100$ starting points uniformly drawn in Ω and iterate the dynamical system for $N_{\max}$ time steps with step size $\Delta_t = 0.1$. If the end-point $\mathbf{x}(N_{\max})$ of the reproduced trajectory is in the vicinity of the desired attractor $\mathbf{x}^* = \mathbf{0}$, the learned dynamics are rated as converged to the target (i.e. $\|\mathbf{x}(N_{\max}) - \mathbf{x}^*\| < \delta = 1$), otherwise as diverged. The distance $\text{dist} = \|\mathbf{x}(N_{\max}) - \mathbf{x}^*\|$ of the end-point to the attractor for each converged trajectory constitutes the second measure. The amount of the converged trajectories S divided by the number of starting points N_s quantifies the stability of the system as a third measure. In addition, the CPU time used for training conducted on a x86_64 linux machine with at least 4 GB of memory and a 2+ GHz intel processor in matlab R2010a[3] is recorded. All results are averaged over 10 different network initializations.

[2]The dataset was collected at the LASA institute at EPFL.

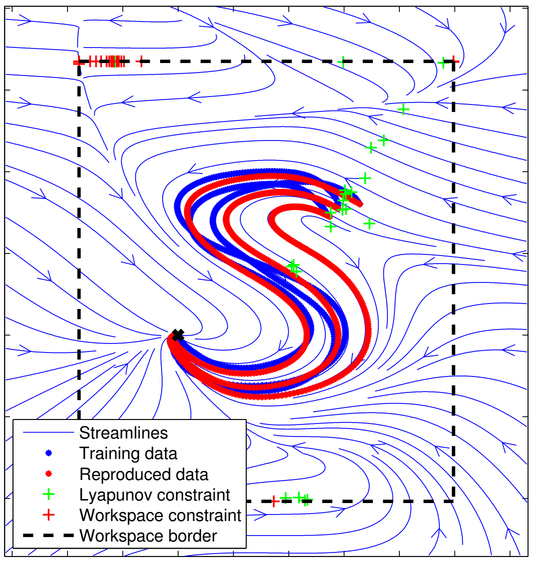
[3]MATLAB. Natick, Massachusetts: The MathWorks Inc., 2010.



(a)



(b)



(c)

Fig. 3.6: The impact of the incorporation of asymptotic stability into the learning. Visualized dynamics of a network trained without stability constraints Fig. 3.6(a) and the same network trained with constraints for stabilization Fig. 3.6(b). A network where constraints are sampled only in a half of the task space Fig. 3.6(c).

The results of the experiments for networks with and without stabilization for different regularization parameters ε are shown in Tab. 3.1. The performance of implementing the desired attractor is given by the Euclidean distance (dist). The ratio S/N_{s} shows how many motions generated according to the discretization of the dynamics are close to the desired target. Note that $S/N_{\mathrm{s}} = 1$ holds for all constrained networks. The results show that the stability ratio increases with increasing regularization ε for networks trained without constraints. This reveals a trade-off between stability and accuracy for the unconstrained networks. For the explicitly stabilized ELMs, the trade-off is resolved, because all networks converge to the target independent of the regularization while the network can still approximate the training data accurately for less regularization. Thus, the target attractor is imprinted close to the desired attractor (cf.

without constraints						
$\lg \varepsilon$	dist	S/N_s	E_{tr}			CPU time [s]
-8	.431±.067	.686±.053	.21±.0006			0.22 ± 0.001
-6	.431±.018	.813±.031	.22±.0008			0.21 ± 0.001
-4	.401±.013	.917±.053	.24±.0027			0.22 ± 0.001
-2	.382±.010	.965±.044	.30±.0028			0.20 ± 0.001

with constraints						
$\lg \varepsilon$	dist	S/N_s	E_{tr}	$\# U_1$	$\# U_2$	CPU time [s]
-8	.049±.014	1	.26 ± .0008	52.1 ± 6.1	47.8 ± 8.4	638.6 ± 184.2
-6	.043±.016	1	.28 ± .0011	32.7 ± 5.9	26.4 ± 4.7	249.1 ± 80.4
-4	.055±.014	1	.31 ± .0014	23.3 ± 4.4	12.4 ± 4.6	139.2 ± 44.9
-2	.020±.012	1	.38 ± .0015	8.5 ± 0.7	13.6 ± 3.6	59.4 ± 18.6

Tab. 3.1: Network learning with and without constraints compared for different regularization parameters ε averaged over 10 trails. Additional information about the learning time and the amount of derived constraints ($\#U_1$) from the Lyapunov candidate L_q and from the workspace constraint ($\#U_2$) is given. S/N_s is the ratio of converged motions.

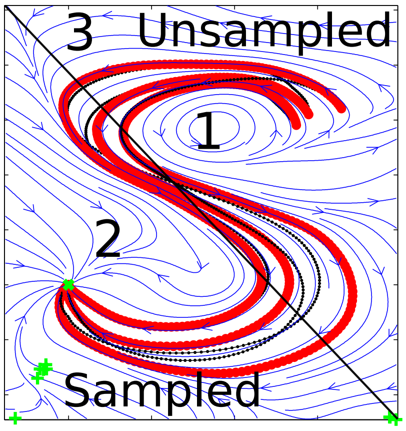


Fig. 3.7: The impact of sampling stability constraints for the learning only in one half of the workspace. The workspace is divided into two sub-spaces where one is subject to constraint sampling (samples are shown in green). Three interesting regions can be seen. (1) illustrates a spiral repeller, (2) stable dynamics and (3) generalized stable dynamics.

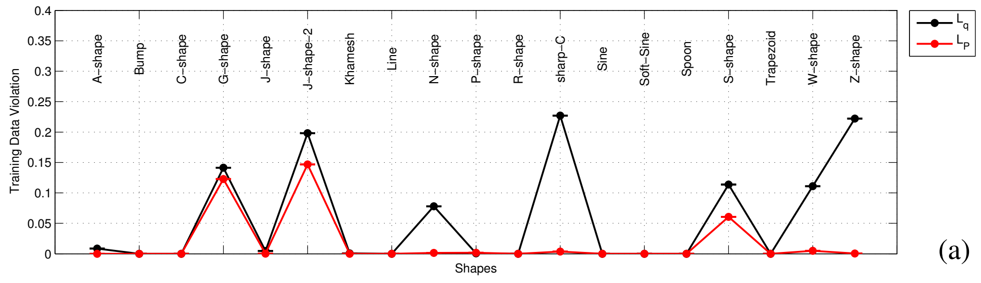
dist in Tab. 3.1).

The successive implementation of the constraints is computationally more costly (this is strongly depending on the number of constraint samples $\#U_1$ and $\#U_2$ that are used for training) but can be reduced by increasing regularization. The main reason for this effect is that the generalization of the constraint becomes easier with lower model complexity (i.e. by increasing ε). Note that the positioning of the samples is non-uniform, because the samples are only placed in regions that are more prone to instability with respect to the Lyapunov candidate and the border of the workspace Ω.

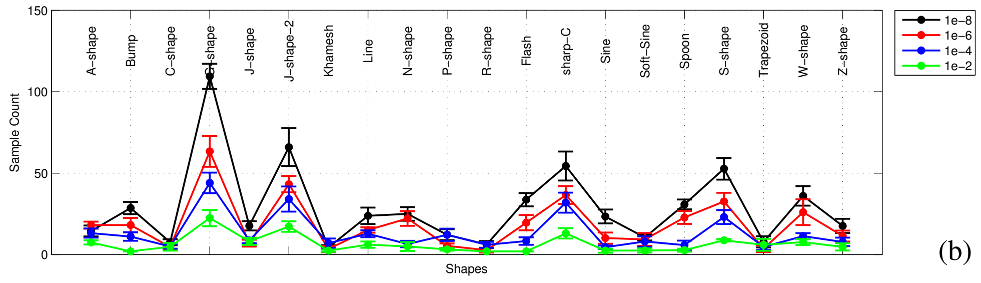
3.5.1 Flexible distribution of stability-constraints

Fig. 3.7 demonstrates the locality of the proposed approach, please compare to Fig. 3.6. In this experiment, only the region below the black stroke is subject to constraint sampling and no border constraint is considered. Three regions in this experiment are interesting: the spiral repeller in region 1 is still active since the data close to this region forces the dynamics to this behavior. The spiral and spurious attractor in region 2, that appears in Fig. 3.6(a), is deleted due to the sampling of constraints. Region 3 shows that the stability constraint can be generalized towards regions not subject to sampling, since no data is present in this region, thanks to the strong model bias induced by the stability constraints.

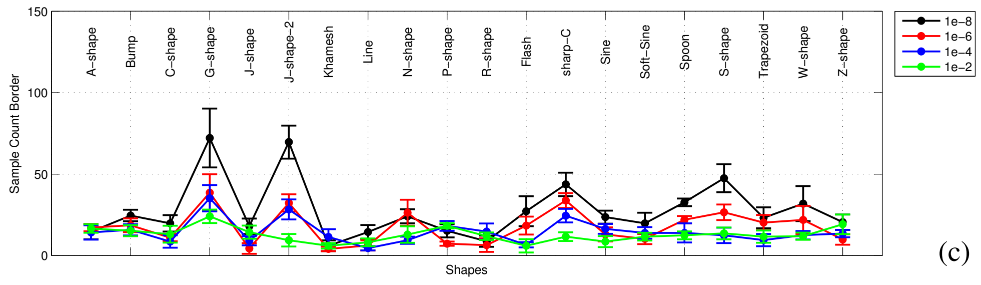
This example shows that the locality of this approach is no disadvantage. Instead, it emphasizes that locality adds high flexibility to the approach: The application of different constraints in selected regions is possible.



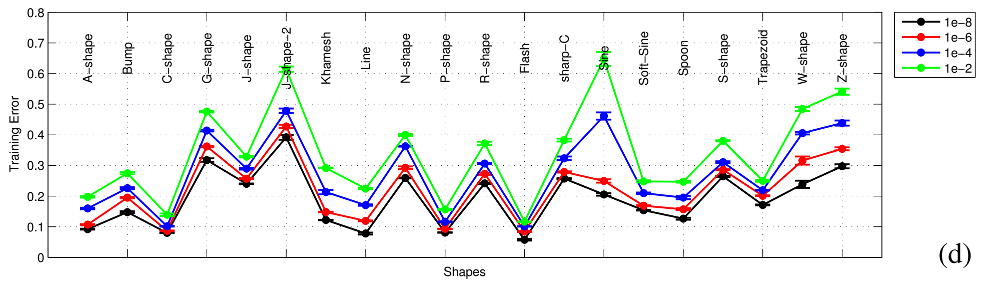
(a)



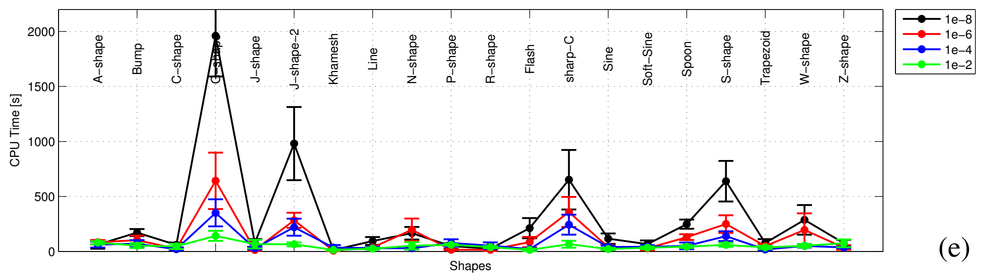
(b)



(c)



(d)



(e)

Fig. 3.8: Investigation on 20 motion shapes of the LASA dataset [Khansari-Zadeh, 2012a]. Evaluation of the violation measure M on each motion (Fig. 3.8(a)) comparing the quadratic Lyapunov candidate L_q with the parameterized version L_P. The following plots show the results using L_P and different regularizations ε. In Fig. 3.8(d) show the the training errors. In Fig. 3.8(b) and Fig. 3.8(c) the amount of samples is shown, used for learning without and with border constraints. At last in Fig. 3.8(e) the CPU time is shown during learning when using L_p with both types of constraints.

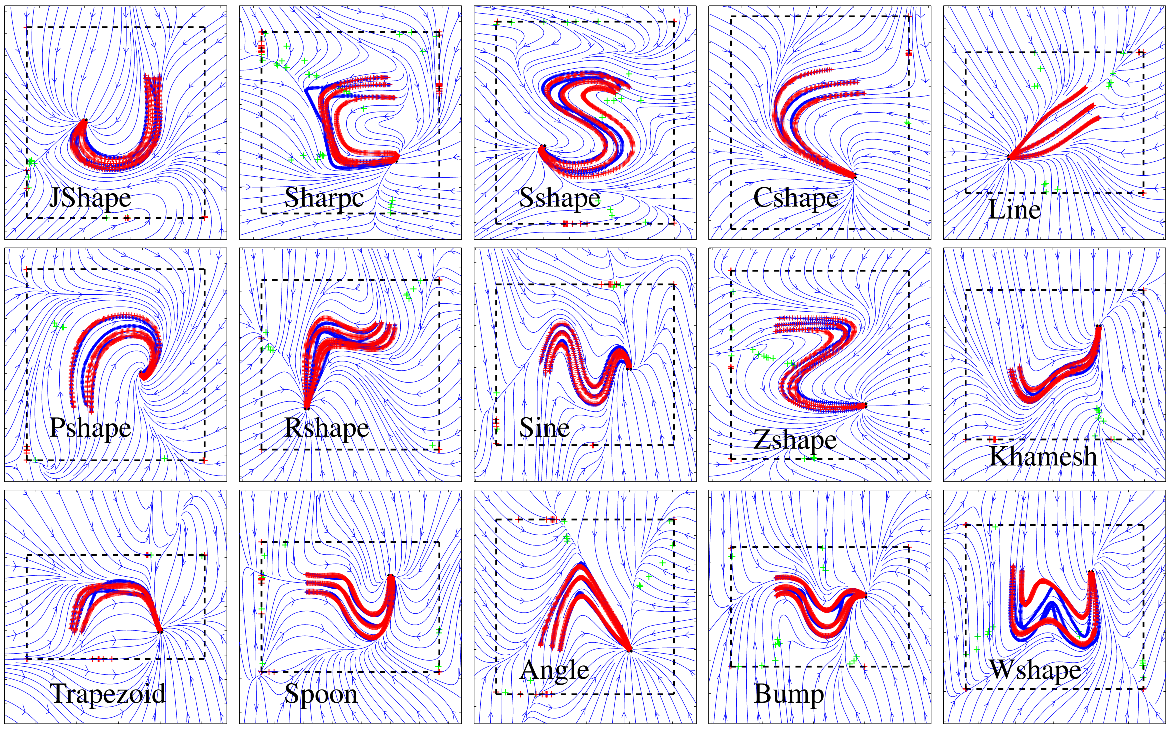


Fig. 3.9: The figure shows 15 examples from the LASA benchmark dataset. The black lines represent the training data and the red lines are the reproduced movements. The learned vector fields are depicted in blue.

3.5.2 Lyapunov candidate versus training data

The experiments described in this section, evaluate the learning process with respect to the used Lyapunov candidate and the training data.

Fig. 3.8 shows an investigation of several measures on each of the twenty motion shapes separately. The plot in the upper part of the figure shows the value for M defined in Eq. (3.17) for each shape of the dataset. It is revealed that the simple quadratic Lyapunov candidate $L_q = \mathbf{x}^2$ strongly violates the training data for some of the shapes (see Fig. 3.8(a)). This violation can be relaxed by introduction of the matrix P. Some of the shapes are not violated to a high degree anymore after optimization - e.g. the sharp-C shape. These results support the idea that optimized Lyapunov candidates enhance the class of accurately learn able shapes. The plot in Fig. 3.8(b) and Fig. 3.8(c) shows the number of samples drawn in the learning phase until implementation of asymptotic stability. The plot summarizes the results for networks with different regularization parameters. Two results can be deduced from this experiment. First, there is a clear correlation between “difficult” shapes - in the sense of M - and the number of applied samples: difficult shapes need more samples for implementation of stability. Second, networks with stronger regularization need less constraints for enforcing stability. This is advantageous, because it significantly reduces the time needed for learning. Note, that the number of samples for some shapes could be reduced significantly by increasing the regularization parameter from $\varepsilon = 10^{-8}$ to $\varepsilon = 10^{-2}$.

Fig. 3.8(d) shows the training error obtained in this experiment. It can be seen that the training error is high for shapes which also produce a high value of violation M. This is expected because the networks are forced to implement the Lyapunov candidate, i.e. the constraints have a higher priority then the data. Thus, if the Lyapunov candidate violates the training data, the network predictions will not accurately reproduce the demonstrations. It is also revealed that the training error is decreasing for lower regularization parameters ε. The additional experiments show that the regularization parameter is still important to tune. However, the stability of the resulting estimate is mainly influenced by the sampling scheme, and consequently also the time needed for training (compare Fig. 3.8(a) and Fig. 3.8(e)).

Fig. 3.9 shows movements learned from demonstrations (black) with the reproduced trajectories

A-shape	Time [s]	M	E_{velo}	E_{traj}
SEDS	22.1 ± 1.3	0.008	$.113 \pm .0029$	$.0168 \pm .0004$
L_q	9.2 ± 0.9	0.008	$.106 \pm .0006$	$.0081 \pm .0003$
L_P	$\mathbf{7.8 \pm 0.8}$	0.000	$\mathbf{.096 \pm .0015}$	$\mathbf{.0053 \pm .0002}$
L_{ELM}	50.5 ± 4.6	0.000	$.103 \pm .0023$	$.0066 \pm .0004$
J-2-shape	Time [s]	M	E_{velo}	E_{traj}
SEDS	$\mathbf{23.0 \pm 1.4}$	0.198	$.119 \pm .0008$	$.0338 \pm .0001$
L_q	49.8 ± 3.5	0.198	$.093 \pm .0015$	$.0298 \pm .0032$
L_P	36.4 ± 2.7	0.147	$.143 \pm .0011$	$.0241 \pm .0004$
L_{ELM}	55.2 ± 4.8	0.000	$\mathbf{.069 \pm .0013}$	$\mathbf{.0079 \pm .0004}$
LASA all	Time [s]	M	E_{velo}	E_{traj}
SEDS	22.6 ± 1.3	0.0582	$.109 \pm .0020$	$.0169 \pm .0003$
L_q	$\mathbf{14.3 \pm 1.5}$	0.0582	$.114 \pm .0003$	$.0218 \pm .0005$
L_P	15.9 ± 2.3	0.0180	$.110 \pm .0004$	$.0177 \pm .0003$
L_{ELM}	52.4 ± 4.0	0.0001	$\mathbf{.103 \pm .0006}$	$\mathbf{.0145 \pm .0004}$

Tab. 3.2: Results for A-shape, J-2-shape, and the entire dataset.

(red) and the surrounding vector field (blue). Note that all networks produce stable and accurate movements which converge to the given target point attractor due to the combination of optimized Lyapunov candidate and constrained learning.

3.5.3 Comparisons of all gained Lyapunov candidates

This section extends the evaluation of the proposed approach to experiments performed on the entire LASA dataset of handwriting motions.

In the previous sections, three different Lyapunov candidates L_q, L_P, and L_{ELM} are suggested for learning. The performance of these candidates in comparison to SEDS[4] are further evaluated on a library of 20 human handwriting motions from the LASA dataset [Khansari-Zadeh, 2012a]. The local accuracy of the estimate is measured according to [Khansari-Zadeh and Billard, 2011]:

$$\begin{aligned} E_{\text{velo}} = \frac{1}{N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \Bigg(r \left(1 - \frac{\mathbf{v}^i(k)^T \hat{\mathbf{v}}(\mathbf{x}^i(k))}{\|\mathbf{v}^i(k)\| \|\hat{\mathbf{v}}(\mathbf{x}^i(k))\| + \varepsilon} \right)^2 \\ + q \frac{(\mathbf{v}^i(k) - \hat{\mathbf{v}}(\mathbf{x}^i(k)))^T (\mathbf{v}^i(k) - \hat{\mathbf{v}}(\mathbf{x}^i(k)))}{\|\mathbf{v}^i(k)\| \|\mathbf{v}^i(k)\| + \varepsilon} \Bigg)^{\frac{1}{2}} , \end{aligned} \tag{3.21}$$

which quantifies the discrepancy between the direction and magnitude of the estimated and demonstrated velocity vectors for all training data points[5]. The accuracy of the reproductions is measured according to:

$$E_{\text{traj}} = \frac{1}{T \cdot N_{\text{ds}}} \sum_{i=1}^{N_{\text{traj}}} \sum_{k=1}^{N^i} \min_l \|\hat{\mathbf{x}}^i(k) - \mathbf{x}^i(l)\| , \tag{3.22}$$

where $\hat{\mathbf{x}}^i(\cdot)$ is the equidistantly sampled reproduction of the trajectory $\mathbf{x}^i(\cdot)$ and T denotes the mean length of the demonstrations. Nevertheless, it is believed that the trajectory error is a more appropriate measure than the velocity error to quantify the quality of the dynamical estimate.

The results for each shape are averaged over 10 network initialization. Each network used for estimation of the dynamical system comprises $R = 100$ hidden layer neurons, $\varepsilon_{\text{RR}} = 10^{-8}$ as regularization

[4]The publicly available SEDS software by Khansari-Zadeh et al. [Khansari-Zadeh, 2013] is used.

[5]Measure and values ($r = 0.6$, $q = 0.4$) taken from [Khansari-Zadeh and Billard, 2011]. The regularization parameter is set to $\varepsilon = 10^{-6}$.

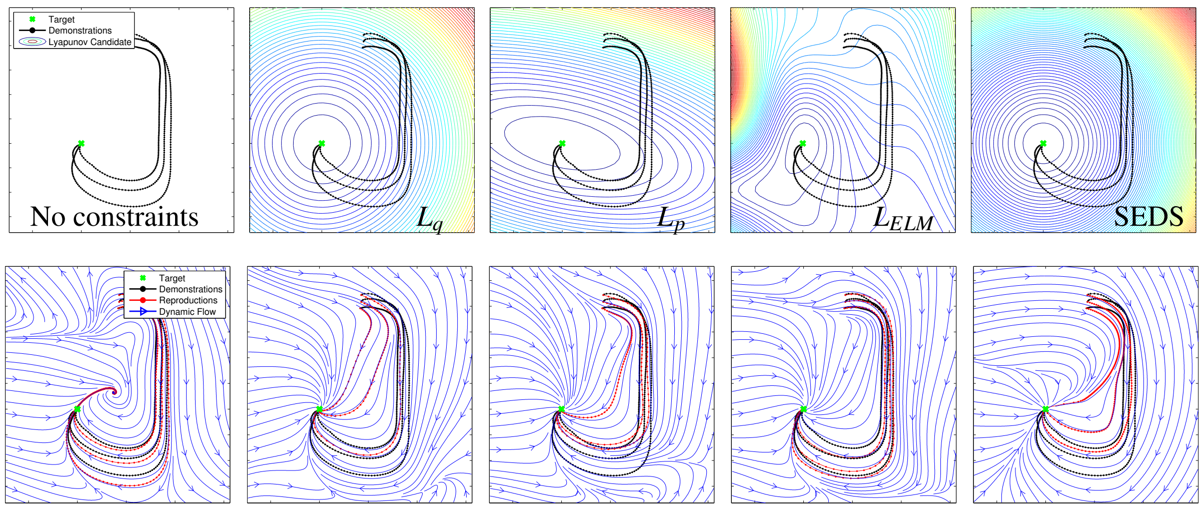


Fig. 3.10: Estimates of the J-2-shape and respective Lyapunov candidates. The J-2-shape approximated without explicit stabilization (first column), with L_q (second column), L_P (third column), L_{ELM} as Lyapunov candidate (fourth column). The SEDS estimate (fifth column, first row). The stability conditions in SEDS are derived based on a quadratic energy function [Khansari-Zadeh, 2012b] (fifth column, second row).

parameter, and $N_C = 10^5$ samples for learning. The networks for Lyapunov candidate learning also comprise $R = 100$ neurons in the hidden layer and $\varepsilon_{\mathrm{RR}} = 10^{-8}$ as regularization parameter. The SEDS models where initialized with $K = 5$ Gaussian functions in the *mse* mode and trained for maximally 500 iterations. Two movements from the dataset are taken to analyze the performance of the methods in detail: A-shape and J-2-shape. This demonstrations are shown in Fig. 3.3 (left) and Fig. 3.10, respectively. The experimental results are provided in Tab. 3.2. The table contains the time used for computation in seconds, the measure of violation M, the velocity error according to Eq. (3.21), and the trajectory error defined in Eq. (3.22). All experiments have been accomplished on a x86_64 linux machine with at least 4 GB of memory and a 2+ GHz intel processor in matlab R2010a. The overall results for the LASA dataset are collected in Tab. 3.2 (last tabular). Fig. 3.10 visualizes the estimation of the J-2-shape and the respective Lyapunov candidates.

The numerical results in Tab. 3.2 show that the function used for implementation of asymptotic stability has a strong impact on the approximation ability of the networks. The A-shape (see first tabular) was accurately learned by all models. The reason is that the demonstrations do not violate the respective Lyapunov candidates to a high degree which is indicated by small value of M. The J-2-shape (see second tabular of Tab. 3.2) is one of the most difficult shapes among the shapes in the LASA dataset. The experiments reveal that the networks trained with L_{ELM} candidate stabilization have the best velocity and trajectory error values. This is due the high flexibility of the candidate function which eliminates the violation of the training data. The third tabular in Tab. 3.2 shows the results for the whole dataset. It shows that the differences in the velocity error are marginal in comparison to the values for the trajectory error. The networks trained with the quadratic Lyapunov candidate L_q perform the worst because the function cannot capture the structure in the data. SEDS performs in the range of the quadratic functions due the conservative stability constraints. The method using L_{ELM} has the lowest trajectory error values which is due to the high flexibility of the candidate function. Methods which cannot avoid the violation of the training data are, in principle, not able to approximate a given dataset accurately while providing stable dynamics. The experiments thus reveal that a flexible, data-dependent construction and application of a Lyapunov candidate is indispensable to resolve the trade-off between stability and accuracy.

In addition to the experiments, Fig. 3.10 shows the estimations of the J-2-shape and their respective Lyapunov candidates. The left column of the figure contains the estimation result obtained for an ELM without any prior knowledge to stability. Obviously, even reproductions starting in the vicinity of

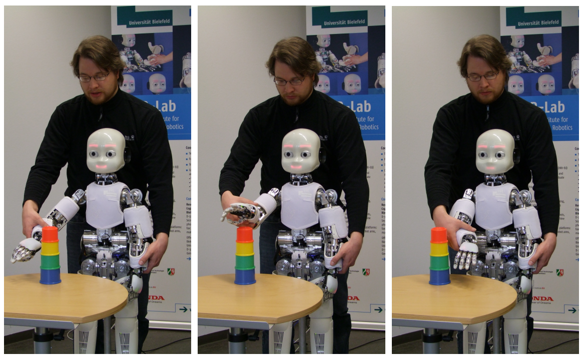


Fig. 3.11: Kinesthetic teaching of iCub. The tutor moves iCub's right arm from the right to the left side of the small colored tower.

the demonstrations are prone to divergence. The second column of Fig. 3.10 illustrates the results for networks trained with respect to L_q. It is shown that this Lyapunov candidate introduces a very strict form of stability, irrespective of the demonstrations. The reproductions are directly converging towards the attractor. This is due to the high violation of the demonstrations by L_q close to the start of the demonstrations. Column three of Fig. 3.10 shows the results for L_P. This Lyapunov candidate is data-dependent but still too limited to capture the full structure of the J-2 demonstrations. The fourth column of the figure illustrates the performance of the networks trained by L_{ELM}. The Lyapunov candidate is strongly curved to follow the demonstrations closely (first row, fourth column). The estimate leads to very accurate reproductions with good generalization capability. The results for SEDS are shown in the fifth column of the figure. As mentioned, SEDS is subject to constraints corresponding to a quadratic Lyapunov function L_q. The results are very similar to the results for the networks applying L_q or L_P as Lyapunov candidate.

3.6 Learning from kinesthetic teaching

The presented Lyapunov approach is applied to a real world scenario involving the humanoid robot iCub [Tsagarakis et al., 2007]. Such robots are typically designed to solve service tasks in environments where a high flexibility is required. Robust adaptability by means of learning is thus a prerequisite for such systems. The experimental setting is illustrated in Fig. 3.11. A human tutor physically guides iCubs right arm in the sense of kinesthetic teaching using a recently established force control on the robot [Fumagalli et al., 2011]. The tutor can thereby actively move all joints of the arm to place the end-effector at the desired position. The tutor first moves the arm, beginning on the right side of the workspace, around the obstacle on the table and touches its top. The guided movement proceeds then towards the left side of the obstacle and stops. This procedure is repeated three times.

The recorded demonstrations comprise between $N_{\mathrm{traj}} = 542$ and $N_{\mathrm{traj}} = 644$ samples. The hidden layer of the networks estimating the dynamical system consists of $R = 100$ neurons and the regression parameter is $\varepsilon_{\mathrm{RR}} = 10^{-5}$ in the experiment as well as for the network used for L_{ELM}. The weights and biases of the network are initialized randomly from a uniform distribution in the interval $[-10, 10]$ due to the low ranges of the movement. The results of the experiment are visualized in Fig. 3.12. The figure Fig. 3.12(a) shows the impact of the different Lyapunov candidates on the estimation of the dynamics. The estimation of the networks trained by the quadratic function L_q is not able to capture the complex shape of the dynamics. The networks trained with the L_P function provides a better performance. The networks using the L_{ELM} function yields a good estimate.

The second plot in Fig. 3.12(b) illustrates the robustness of the learned dynamics against spatial perturbations. Therefore, $N = 75$ starting points are randomly drawn from a uniform distribution in $\Omega = [-0.05, 0.1] \times [-0.5, 0.25] \times [-0.05, 0.2]$. Even the trajectories which start far away from the demonstrations converge to the target attractor and thus underline the robustness of the proposed learn-

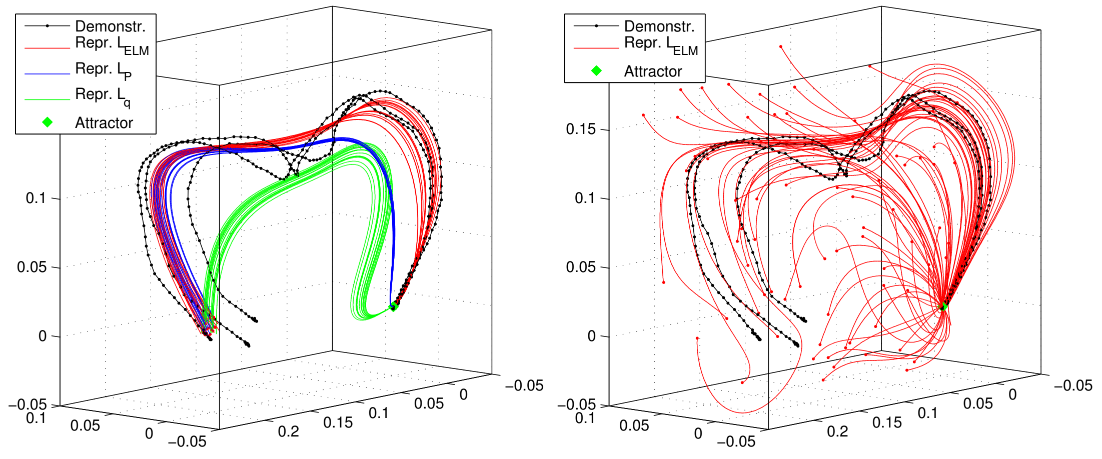


Fig. 3.12: Results of the iCub teaching experiment. Reproduction of the learned trajectories in meter according to the Lyapunov candidates L_q, L_P, and L_{ELM} (Fig. 3.12(a)). Reproductions according to L_{ELM} subject to random spatial perturbations (Fig. 3.12(b)).

ing method.

In Tab. 3.3 the important parameters are summarizes, which are set before the learning process starts. Note that only three additional parameters are added (top) to the standard ELM parameters (bottom). The Lyapunov candidate L is the most important variable to reach good performance in approximating the training data. The size of the workspace is mostly task dependent and needs to be determined accordingly. The amount of samples N_C drawn from this region is a crucial parameter with respect to computation time, but does not need much tuning. If too few samples are drawn, the probability to implement asymptotic stability decreases, because regions for which the learner violates the stability constraints maybe overlooked. Thus, this parameter needs to be linked to the size of the workspace Ω.

Important parameters for learning	
L	Suitable Lyapunov candidate to draw constraints from
Ω	Workspace to implement constraints
N_C	Amount of samples drawn from region Ω used in Alg. 3.1
ε	Regression parameter
R	Number of neurons in the hidden layer
$\mathbf{W}^{inp}$	Initialization range of the input weight matrix (input scaling)
$\mathbf{a}, \mathbf{b}$	Initialization ranges of the nonlinearity see Eq. (3.5)

Tab. 3.3: Important parameters which need to be set by the user before network learning. The last set of parameters are needed already for the standard ELM approach. The first part of the table gives the new parameters important for inducing stability.

3.7 Discussion

The learning methodology described in this chapter ensures that the adapted dynamical systems are asymptotically stable in a predefined local workspace. The main advantage of the proposed approach is the inherently decoupled process of finding a proper Lyapunov candidate and implementing the dynamical systems itself. In principal this allows the implementation of arbitrary Lyapunov candidates and can thus lead to flexible, robust, and accurate dynamical systems learned from sparse data. However, to allow a good performance in approximating the training data, it is essential that the Lyapunov function does not contradict the training data. Therefore, finding a suitable Lyapunov candidate is one of the key steps in this approach. The standard quadratic energy functions are for many motions too restrictive and results in poor reproduction of the targeted dynamics.

The learning process uses constraints that are generated by a sampling strategy. The proposed sampling strategy finds potential unstable regions and uses them to constructively build stability into the adapted dynamics. The locality of the approach permits to choose desired regions for the implementation of constraints. This flexibility is advantageous if a variety of such constraints has to be satisfied. They can appear in several bounded regions in possibly lower dimensional sub-regions of the input space. They can be defined only at certain discrete points or in continuous regions. Also, multiple attractors can be implemented with this approach. Although, the locality of this approach prevents a direct stability guarantee for a previously defined basin of attraction, asymptotic stability can be effectively proven ex-post, if needed. This is possible, because the neural network allows analytically differentiation. This approach exploits the linear dependency on the learning parameters and uses a worst case approximation by means of Taylor polynomials (see [Neumann et al., 2013b]).

Scaling the approach to higher dimensions depends on the number of samples needed to implement stability. The presented results show that the required number of samples and thus the time for training can be significantly reduced by increasing the regularization parameter of the ELM. Although, one cannot limitless increase the regularization parameter without reducing the approximation ability which results in a trade-off between accuracy and computational time. It relies on the sampling-based evaluation of stability constraints which makes the system computationally intractable in higher dimensions.

The guarantee of *global* asymptotic stability is an important point in related approaches. The main advantage of global stability is that the size of the workspace is unconstrained. However, the relevant region of the task space in which the dynamical system shall be able to generalize and perform motions is in many applications bounded and known in advance. In case of a strong perturbation, which pushes the state of the dynamical system outside of the stable region, a high-level controller can react and apply counter measures (e.g. emergency stop of the robot or switch to another motion module). Another possibility is to detect the current position outside of the local region and project the position to the border of the workspace. All velocities on the border are pointing inside of the workspace according to Eq. (3.12). Using this approach the stability can be generalized to the global region, because the constraints defined by the Lyapunov theory, described in Sec. 3.3.1, hold for all positions.

3.8 Concluding remarks

In this chapter a novel learning approach is proposed to learn vector fields from sparse data for motion generation. Stability assumptions inspired from Lyapunov's stability theory are exploited to shape the vector fields. Due to this strong model bias the generalization abilities are enhanced and stable motion generation becomes possible.

In extensive evaluations in this chapter it is shown that the new learning approach is sufficient to implement asymptotic stability from only a few trajectory demonstrations which leads to stable dynamical systems. The learning scheme principally incorporates linear inequality constraints in discrete points of the workspace derived from a so-called Lyapunov candidate efficiently by quadratic programming. Despite the fact that the constraints are only satisfied in discrete points by construction, they can be generalized towards a continuous region by using a sophisticated sampling strategy. This sampling strategy permits to choose desired regions for the implementation of constraints because of the locality of the approach. This flexibility can be advantageous due to the large variety of such constraints. They can appear in several bounded regions in possibly lower dimensional subregions of the input space, they can be defined only at certain discrete points or in continuous regions.

Additional experiments demonstrated the role of regularization in relation to the number of samples needed for implementation of the constraints and in relation to the obtained approximation performance. The accuracy of the estimates were enhanced by using a more flexible Lyapunov candidate function which were adapted to the training data. This new form of Lyapunov candidates relaxed the constraints such that more datasets could be learned with adequate accuracy. Interestingly, it is possible to implement even more complex candidate functions which might be superior to the already used functions. To tackle this question is left open for future work. Finally, it was demonstrated that the learning scheme can cope with data obtained by kinesthetic teaching of the humanoid robot iCub and that it generates smooth and accurate reproductions of the learned demonstrations in a three-dimensional task.

The learning approach is able to estimate vector fields applied in a autonomous dynamical system for movement generation. The method is based on the idea of separating the learning into two main steps: i) learning of a highly flexible Lyapunov candidate from data and ii) implementation of asymptotic stability by means of this obtained function. This approach strongly reduces the trade-off between stability and accuracy, which allows robust and complex movement generation in robotics.

CHAPTER 4

COMPARISONS OF MOVEMENTS UNDER BENCHMARK CONDITIONS

In Chapter 3 a possible learning approach is introduced to create dynamical systems suited for representing movement primitives. It is one example out of many movement primitive representations that have been introduced within the past decade (see Sec. 2.2). Each modeling approach has its own strengths and weaknesses, which can be reliably identified with systematic comparisons. To provide systematic comparisons standardized evaluation and tasks are essential.

In this chapter a benchmark framework and methodologies for systematically testing and evaluation of different movement primitive representations are described. The design of the benchmark is highly influenced by the advent of a new generation of humanoid robots that need to perform a wide variety of tasks in human daily lives. It is essential that humanoid robots can move similar to human beings for social acceptance as discussed e.g. in [Oztop et al., 2005; Chaminade et al., 2005]. However, from the robotics perspective each movement primitive need to have motion generation techniques that can cope autonomously with changing situations. These situations may be corrupted with various sources of perturbations and other uncertainties. Therefor, two main criteria need to be evaluated in this benchmark which is the *human-likeness* of the motion generation and also the *robustness* of the motion generation.

Participation in the proposed benchmark is a valuable addition to the research in this field and findings could lead to greater understanding of the differences in the various movement primitive representations. At [Khansari et al., 2013], this benchmark framework was first introduced and discussed, where first benchmark users presented their preliminary results. The conceptual ideas are also published in [Lemme et al., 2015].

4.1 Design of a systematic comparison

In general, designing benchmarks is a common approach for providing systematic comparisons in many research fields, e.g. supercomputers [Bailey and Barton, 1985], hardware design [Gaj et al., 2010], or optimization software [Dolan and More, 2002]. This motivates to propose a benchmark software framework, where motion generation algorithms are considered that model point-to-point reaching or drawing motions with the help of dynamical systems as discussed in Sec. 2.2.

In most previous works regarding movement primitives, an extensive performance and generalization evaluation is missing. Success is illustrated mostly by displaying only a few number of example motions. However, there is a large number of possible training datasets, generalization tasks and measures that could potentially be useful for evaluation. This provides a substantial challenge for designing a respective benchmark. It needs to provide a rich set of training-data and to evaluate a specific, standardized set of performance measures on significant tasks. These tasks should be reasonably diverse and challenging. In particular, it is not sufficient to evaluate only the accuracy of reproducing a given (set of) demonstrations. On the contrary, for actual applications on real robots, it is essential to evaluate motion generation methods on how robust they are against disturbances and uncertainties. Therefore, disturbances or uncertainties like e.g. 'perturbations applied to the end-effector' or 'changes in the goal state' are applied during the execution of motions in the described benchmark.

The decision on which performance measures are used depends on the desired properties for motion generation. In this chapter, it is argued that robots must be (i) precise in the execution of the task, (ii) robust to perturbations, which may occur during execution and (iii) robots need to move similar to humans to improve the acceptance of robots in human environments. Following this argumentation, the benchmark framework comprises of ten different evaluation criteria. It includes measures derived from the investigation of human motions in order to measure human-likeness using typical features like power laws [Viviani and Cenzato, 1985] or the minimum jerk model [Hogan, 1984; Flash and Hogan, 1985; Todorov and Jordan, 1998]. Whereas the control principles underlying these regularities are disputed in movement science [Flash et al., 2013], this work does not emphasize a particular theory.

The idea is to propose measures that evaluate particular features, strength or respective weaknesses and seek to develop a multi-faceted benchmark system. Consequently, the benchmark offers means for statistical evaluation and many diverse measures. Due to the fact that the focus of each user is maybe different, it is refrained to provide an automatically generated single number for the performance ranking of different methods and leaves the ranking to the user. This benchmark framework considers only the motion generation of already *trained modules*, therefore, does not consider the learning process during training (e.g. time needed for learning etc.).

The contribution of this chapter is to provide a benchmark software framework, which allows to execute a standardized testing for systematic comparisons of motion generation models that are able to learn from demonstrations. Evaluation methods from the robotics and the motor control community are utilized to evaluate precision of execution and human-likeness of the delivered performance of each movement primitive approach.

4.2 Benchmark system for human-like motion generation methods

In this benchmark system, the motion generation task is to reproduce learned point-to-point motions under occurrence of perturbations. In this section an overview of the benchmark system is given. The benchmark system according to Fig. 4.1 is described to clarify what is delivered by the software framework and what needs to be delivered by the participant.

Participants need to prepare their algorithms according to a software interface and a training dataset available in this framework. In the benchmark system, it is assumed that a prescribed set of trained models (see below) is provided corresponding to each motion in the considered dataset.

Trained modules are represented in a class structure, which holds the learned representation of a movement and the control algorithm. This provides the functionality on how to process the trained modules. In the software module interface it is required that each module is represented as a first order dynamical system mapping positions $\mathbf{x}$ to velocity vectors $\mathbf{v}$. Motion generation is conducted by integrating velocities over time t:

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta t \cdot \mathbf{v}(\mathbf{x}^t), \tag{4.1}$$

where Δt is a time constant for discretization of the continuous dynamics and is set according to the ground truth dataset.

Datasets are an important factor in context of this benchmark, since it presents the ground-truth for comparisons and provides the initial conditions for the motion generation schemes. The initial conditions are given by the start points and target points from each demonstrated trajectory. Also the parameter Δt is extracted from the dataset. A dataset is provided in this benchmark and is described in Sec. 4.3.

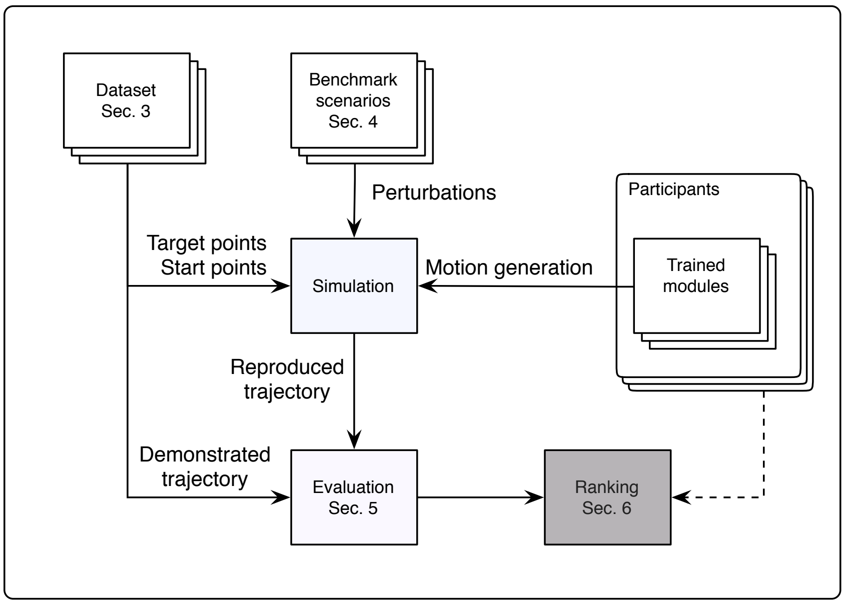


Fig. 4.1: Schematic illustration of the benchmark architecture logic. A default 'Dataset' and 'Benchmark scenarios' are provided. According to the used 'Dataset' the participants need to provide 'Trained modules'. The core of this benchmark system is the 'Simulation' function. It combines all information about initial conditions and uncertainties of the perturbations and executes the 'Trained modules' in the test environment. The 'Evaluation' is evaluating the reproduced trajectories according to the defined measures and the demonstrated trajectories which are the baseline for the systematic comparisons. The results of the evaluations can be utilized by each participant to produce a ranking using a combination of scores.

Benchmark scenarios specify the uncertainties and perturbation types, which can occur during motion generation (e.g. a displacement of the predicted position). An overview of the available scenarios is given in Sec. 4.4.

The simulation function receives the initial start and target points from a given dataset and the perturbation specifications for each benchmark scenario. Given these initial start conditions, the trained modules should generate a trajectory to a given target point. The system provides feedback on the current position of the target and end-effector, the current velocity and the current time step. The simulation stops if either the module indicates that the motion generation is finished or if the generation process exceeds a maximum execution time allowed for the motion.

The evaluation is done by a standard set of measures which identify properties from the delivered reproduced trajectory and are used to compare it to the demonstrated trajectory given by the ground-truth dataset. The provided measures are introduced in Sec. 4.5.

The ranking , if multiple measures are used, is a difficult task, especially, if the focus is not completely known. The result of this benchmark is a variable set of statistical evaluations. This allows for each participant to chose between the evaluations and generate a solid basis for the comparison of his interest. This issue is further elaborated in Sec. 4.6.

4.3 A dataset of human handwriting demonstrations

The ability to encode a wide variety of movement shapes is an essential feature of the considered movement primitive modules. For this purpose, the proposed benchmark uses the LASA human handwriting library [Khansari-Zadeh, 2012a] as a benchmark dataset. This library was first introduced in [Khansari-Zadeh and Billard, 2010a,b] and extended in [Khansari-Zadeh, 2012b; Khansari-Zadeh and Billard,

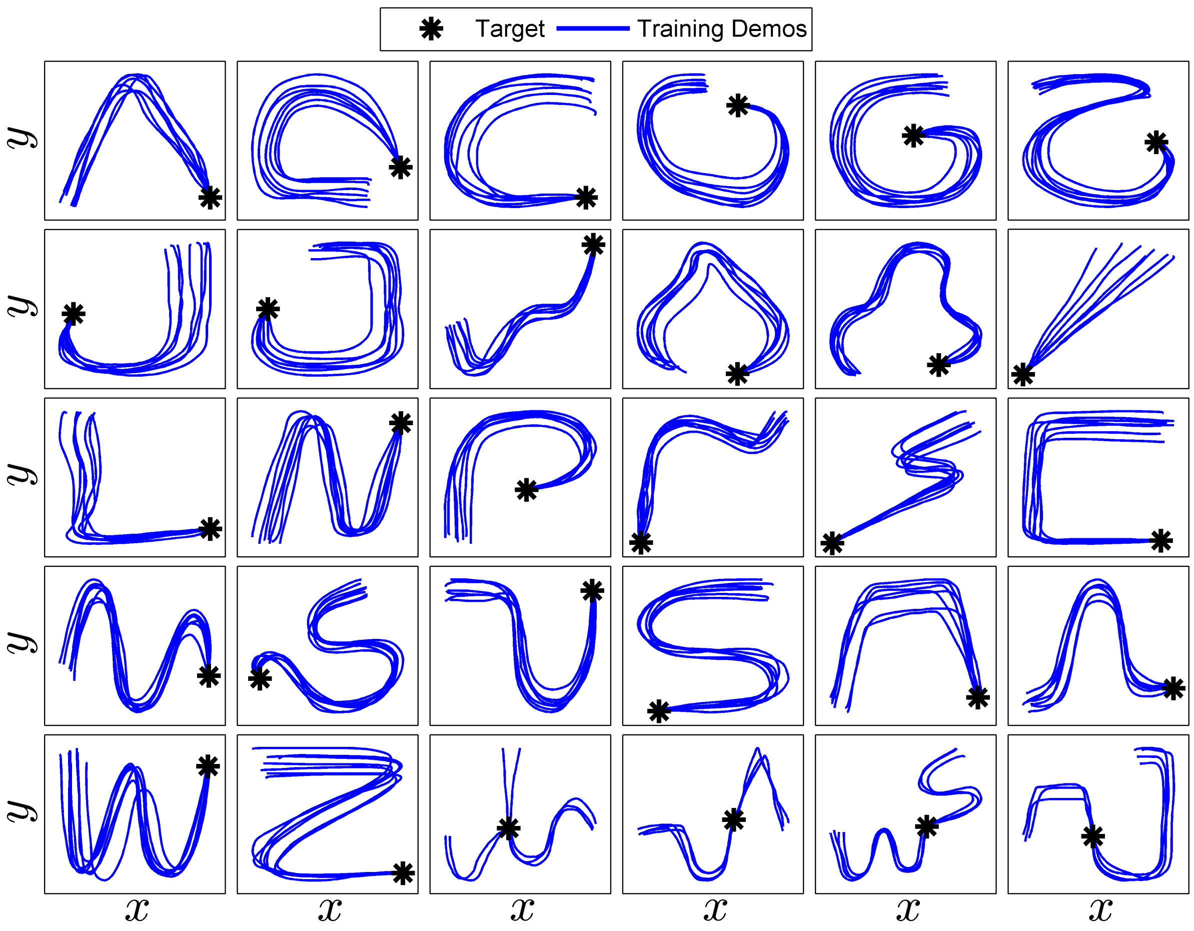
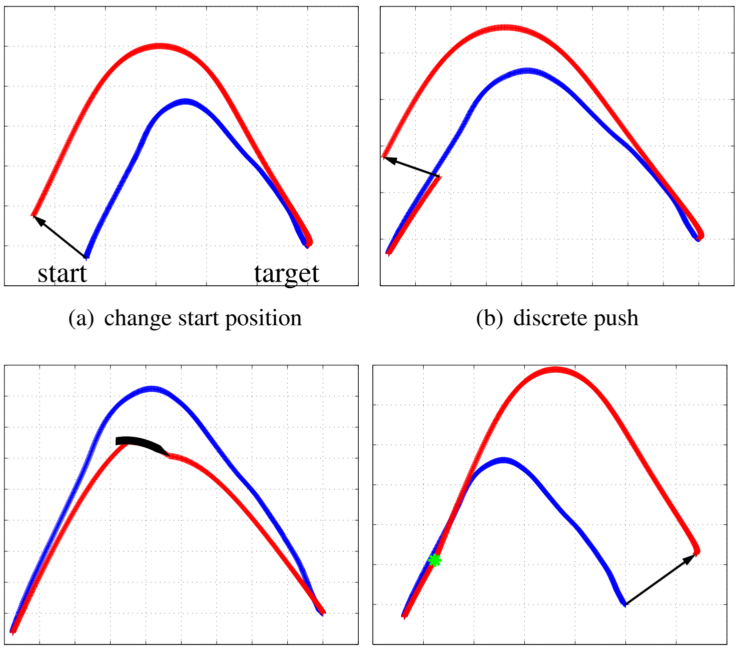


Fig. 4.2: The library of LASA handwriting motions [Khansari-Zadeh, 2012a]. This library is composed of 30 two-dimensional point-to-point motions.

2014] to compare the reproduction performance of different regression techniques. It has also been adopted in several works as the baseline for performance comparison [Neumann et al., 2013a; Lemme et al., 2014a; Gómez et al., 2012] and comprised of data from handwriting motions collected from pen input using a Tablet-PC. For each desired movement shape, the human subject was asked to draw seven demonstrations of each shape, by starting from different initial positions and to the same final point. The initial points are close to each other, which results in demonstrations that may intersect each other but represent the desired movement shape with approximately the same size and rotation. The recorded library contains 26 sets of human handwriting motion and four additional sets. The additional sets include more than one movement shape in one set (called Multi Models). Without losing generality, the target (final) point is by definition set to $(0,0)$ for all movement (shapes) in this library. All demonstrations of the different shapes are presented in Fig. 4.2. In particular, this dataset displays a large variety of motions, which represents suitable ground-truth data for the benchmark evaluation for precision criteria and human-likeness.

One major question is what is the right response to perturbations? Due to the property of human-likeness it is expected from the movement primitive modules to react to perturbations in a similar way as humans do. The performance of human subjects under perturbations were examined before [Shadmehr and Mussa-Ivaldi, 1994; Gandolfo et al., 1996; Conditt et al., 1997; Karniel and Mussa-Ivaldi, 2002] and could also be included in this benchmark. However, in these studies the considered tasks are to generate point-to-point straight motions, which are not very challenging for motion learning, but the results could provide interesting insights of each motion generation approach regarding perturbation handling compared to human performances. At this point, however, the evaluation process is simplified



(a) change start position (b) discrete push

(c) continuous push (d) change target position

Fig. 4.3: Four different benchmark scenarios used in the benchmark software framework to test the generalization capabilities. The blue trajectory is the demonstrated motion, whereas the red trajectory is the perturbed motion generated in the simulation. The black arrow indicate the direction and amplitude of the applied perturbation. In Fig. 4.3(a), the initial start point is changed, which is a standard generalization test. In Fig. 4.3(b), a sudden displacement of the current position is simulated. A continuous change to the current position is shown in Fig. 4.3(c), which simulates a guiding of the end-effector of the robot by e.g. a human teacher. In Fig. 4.3(d), the target jumps in one direction at some point in time during the motion generation (indicated by the green '*').

and adhere to the robotic side, by deciding that the perturbation should be compensated for w.r.t. the different criteria.

4.4 Benchmark scenarios

In this benchmark, the motion generation is done under benchmark scenarios where different perturbations are applied. Four scenarios are identified that specify the majority of perturbation types that may occur during motion generation. They are used to evaluate the ability of the different motion generation algorithms to cope with perturbations during motion execution and show their generalization capabilities: (i) Initialization with different starting points, which is the standard generalization test; (ii) push of the end-effector, realized as sudden displacement of the current position; (iii) continuous push of the end-effector; (iv) changes of goal position during the motion execution. These four perturbation types are visualized in Fig. 4.3.

4.4.1 Generalization to different initial conditions

The initial starting point of the motion generation is perturbed (see Fig. 4.3(a)), which is the most common generalization test in this field. The displacement of the end-effector is applied at time $t = 0$ with amplitude a and direction v, where the systematic variations use combinations a, v of the parameters as specified below:

Amplitude: Consider the length span of motion l along both x and y axes. Then, the amplitude of the perturbation a can be drawn from a normal probability distribution according to:

$$a = \mathcal{N}(\mu, \sigma), \tag{4.2}$$

where $\mu = 0.1 \ \ l$ and $\sigma = 0.05 \ \ l$.

Direction: The direction of the perturbation v_p. The vector $v \in \mathbb{R}^n$ is drawn from a uniform distribution in interval $[-0.5, 0.5] \in \mathbb{N}$ to determine a random direction where the vector is pointing at. This vector is normalized and multiplied by the amplitude to obtain the actual perturbation vector $v_p = a\frac{v}{||v||}$.

4.4.2 Discrete push

In this benchmark scenario, a sudden displacement of the current position is applied during motion generation (see Fig. 4.3(b)). This simulates a hit against the end-effector at a particular point in time resulting in a displacement of the current position of the end-effector. This perturbation appears with varying timing t_p, direction v and amplitude a. The start and target points remain fixed.

Direction and amplitude are chosen as described before in Sec. 4.4.1.

Timing: It specifies the point in time t_p when the trajectory will be perturbed. The benchmark dataset consists of multiple motion shapes, where each shape is demonstrated multiple times. For each set of demonstrations the mean motion duration τ is calculated. The perturbation start time t_p is then given by $t_p = t_s \tau$, where t_s is drawn uniformly in $[0, 1] \in \mathbb{N}$.

4.4.3 Continuous push

In this benchmark scenario (see Fig. 4.3(c)), motion generation is continuously perturbed during a specific time interval. This simulates e.g. a teaching scenario in which a human tutor is correcting the movement for a certain duration. As previously, perturbations appear at varying times, directions and amplitudes. The target point remains fixed.

Timing and direction are chosen as described in Sec. 4.4.2.

Amplitude: The amplitude of the perturbation (in millimeters/second). The samples for amplitude are drawn from a normal probability distribution (see Eq. (4.2)) with mean $\mu = 0.5\bar{v}$ and standard deviation $\sigma = 0.25\bar{v}$. The parameter $\bar{v}$ corresponds to the mean speed of all demonstrations across all motions in the library.

Duration: The duration of the perturbation τ_d (in seconds) is given by $\tau_d = t_d \tau$, where t_d is drawn from a uniform distribution in $[0.1, 0.3] \in \mathbb{N}$. τ is again the mean motion duration given by each set of demonstrations per motion shape.

4.4.4 Target displacement

The ability of the trajectory generators to track and reach moving targets (see Fig. 4.3(d)) is quantified by this benchmark scenario. The target motion starts with different timing and lasts over a certain duration. Also the amplitude and direction of the target motion can be changed.

The parameters are equally chosen as in the *Continuous Push* benchmark only applied to the target point. The same probability distributions are applied.

4.4.5 Possible extensions

Four different types of perturbations are implemented in these benchmark scenarios, but one could think of more sophisticated scenarios like e.g. obstacle avoidance. This scenario can quantify the ability of the motion generation approach to avoid obstacles as smooth as possible without colliding with the obstacle. However, this would require an additional mechanism to respond to the obstacle in a meaningful way, e.g. move around the object without contact with a smooth trajectory.

A possible extension to spatial perturbations is to also perturb the inner clock of the benchmark since some motion generation methods are using an explicit representation of time. It is of course difficult to guaranty that the user is using the clock of the system and not an internal clock inside of the model. A possible implementation of such a perturbation is to add noise to the Δt variable to simulate signal delays from the sensor array, which again effectively results in a spatial displacement of the predicted position.

Further investigations are required in order to provide systematic means for performance evaluation in the above four scenarios, and is thus left for possible future extensions.

4.5 Evaluation and performance measures

In the following the evaluation of the perturbed reproduced trajectories is described. All trajectories are scaled in time by a parameter τ ($d\tau \propto dt$) in order to have standardized duration. This allows to inspect motion kinematics independently of the total movement duration. The geometric and kinematic accuracy of the reproductions are inspected separately and in combination. To retain only the shape relevant geometrical information called *path*, all trajectories are parametrized according to their Euclidean arc length, i.e. the result is a resampled motion trajectory with constant speed which provides only the path information of the motion. The applied measures are described in the following.

4.5.1 Measures on the geometric level

In this section, two measures that are applied to evaluate the geometric features are shown.

Path accuracy

The distance between the reproduced path ω_{repro} and demonstrated path ω_{demo} is measured by either the R^2 measure Eq. (4.7) or the point wise root mean squared error (RMSE).

$$RMSE_{pos} = \sqrt{\frac{1}{M}\sum_i ||\omega_{demo}(i) - \omega_{repro}(i)||^2}, \tag{4.3}$$

where $||\cdot||$ specifies the L2-norm and M is the number of discrete points in each path.

Perturbed movements cannot deliver a perfect match, however, it is still interesting to evaluate how close the methods can stay to the demonstrated path.

Target position error

The target position error is necessary to evaluate the accuracy performance, because some robotic tasks have the necessity of high precision e.g. pic and place tasks. The precision in reaching the target point is given by the distance of the last recorded position to the target. Due to robotic tasks such as pick and place of objects, it is interesting to evaluate how precise the end point is implemented in the movement primitive modules.

4.5.2 Measures on the kinematic level

The following measures are specialized to evaluate the performance on velocity or speed profiles.

Velocity accuracy

The reproduced and demonstrated velocity vector profiles are compared using the RMSE:

$$RMSE_{vel} = \sqrt{\frac{1}{M}\sum_i ||v_{demo}(i) - v_{repro}(i)||^2}, \tag{4.4}$$

where v_{demo} and v_{repro} is the first derivative of the trajectories x_{demo} and x_{repro} with length M.

Speed accuracy

The speed profiles of the demonstrated and reproduced trajectories with standardized duration are compared using the R^2 measure,

$$R^2_{\text{speed}} = 1 - \frac{\sum_i (||v_{demo}(i)|| - ||v_{repro}(i)||)^2}{\sum_i (||v_{demo}(i)|| - ||\bar{v}_{demo}||)}, \tag{4.5}$$

where $||\bar{v}_{demo}||$ is the standardized speed profile (e.g. [Sternad and Schaal, 1999; Pollick et al., 2009; Bennequin et al., 2009]).

The R^2 measure compares the predictions of the trained movement generation model against those of the simple model of constant speed motion, where a value of $R^2 = 1$ indicates a perfect agreement. The constant speed model is better than the reproductions at explaining the data of the demonstrations if $R^2 \leq 0$. In the evaluation process the R^2 measure is not only used for the speed profiles, but also e.g. scoring the reproduced paths and trajectories of motion.

Target velocity error

This is a task relevant measure, because in reaching tasks stopping at the target is necessary for safe human robot interaction. Therefore, it is checked if the last velocity generated by the model is close to zero taking the L2-norm of the velocity vector.

Movement duration

The movement duration of the reproduction and the corresponding demonstration are compared. Let us denote the movement duration of the demonstration and the movement duration of its reproduction by t_d^f and t_r^f, respectively. Then the movement duration error is computed according to:

$$\varepsilon_{movement-duration}(t_d^f, t_r^f) = |1.0 - \frac{t_r^f}{t_d^f}| \tag{4.6}$$

This measure is difficult to interpret if perturbations occur during the motion, but it delivers still interesting information about the motion generation together with other measures. For example if the method does need long to reach the target due to accuracy constraints then the motion generation will take significantly longer than it was demonstrated.

4.5.3 Measures using geometric and kinematic levels

In the motor control community laws of motions are observed and extensively discussed, but it is not clear what drives the human motor system per se. This work does subscribe to any theory that claims to know what specific laws of motion are used. However, it is shown that several of these regularities tend to agree with optimal performances for human drawing like motions.

Ample research has been devoted towards the tendency of humans to produce stereotypical motor behavior [Lashley, 1951; Bernstein, 1967; Abend et al., 1982; Flash, 1983; Hogan, 1984; Harris and

Wolpert, 1998; Mussa-Ivaladi and E., 2000; Bizzi et al., 2000; Flash and Hochner, 2005]. One key observation is that point-to-point motions tend to be straight and their speed profiles bell-shaped regardless of the direction and end-point locations of the generated trajectories. A theoretical account for this invariance is suggested by the minimum jerk model [Hogan, 1984; Flash and Hogan, 1985]. The predictions of this model were originally tested for tasks involving via points [Flash and Hogan, 1985] and also curved movements (see the constrained minimum jerk model [Todorov and Jordan, 1998]).

Another regularity is that hand movements tend to slow down if the shape of the trajectory becomes curved. This tendency is quantified by the two-thirds power law [Lacquaniti et al., 1983], which predicts the hand's speed to be proportional to the curvature of the path of the movement raised to the power of minus one third (see Eq. (4.9)). Numerous studies have analyzed the persistence of this rule, mainly in drawing motions [Viviani and Cenzato, 1985] but also for other modalities such as eye pursuits [Viviani and deSperati, 1997]. In this evaluation measures are applied that evaluate the geometrical and also the kinematic information of the reproduction versus the demonstration.

Trajectory accuracy

The trajectories, x_{repro}, x_{demo}, are compared using the following R^2 measure,

$$R^2_{\text{trajectory}} = 1 - \frac{\sum_i (x_{demo}(i) - x_{repro}(i))^2}{\sum_i (x_{demo}(i) - \bar{x}_{demo})}, \tag{4.7}$$

Perturbed trajectories cannot be deliver a perfect match, however, it is still interesting to evaluate how close the methods can stay to the demonstrated trajectory.

Minimum jerk trajectories

The minimum jerk model predicts that human trajectories minimize the following functional,

$$I(r) = \int_0^T \left| \frac{d^3}{dt^3} r \right|^2 dt,$$

where $r(t)$ is any end-effector's trajectory. Then the root of the mean squared derivative (RMSD) of a trajectory $r(t)$ is determined,

$$RMSD(r) = \sqrt{\frac{1}{T} \int_0^T \left| \frac{d^3}{dt^3} r \right|^2 dt}, \tag{4.8}$$

To use this measure also for the perturbed trajectory is not directly possible, because the perturbation apply jerk to the movement. The time when the perturbation occur is known, which allows to ignore this part of the trajectory for this evaluation.

The 2/3 power law

The 2/3 power law predicts the speed profile of a trajectory, $\frac{ds}{dt}$, based on its curvature $\kappa(t)$,

$$\frac{ds}{dt} = \alpha \kappa(t)^{\beta}, \tag{4.9}$$

where α and β are segment-wise constants. The parameter β is usually close to a value of $-\frac{1}{3}$ for drawing motions and α is a gain factor (see [Endres et al., 2013] for the use of this law for complex 3D tasks).

The curvature-speed power laws defined by Eq. (4.9) are prominent features of human drawing movements. The procedure described in this section focuses only on the average compliance with the law rather than on the temporal features of this compliance. The temporal features could be used to infer movement segments as discussed in [Sternad and Schaal, 1999], however, the question of human segmented control is not addressed in this chapter.

To avoid explicit segmentation, sliding windows are used for each demonstration by regressing the R^2 measure for the following linear relation between the logarithm of speed and logarithm of curvature [Viviani and Cenzato, 1985],

$$\log \frac{ds}{dt} = \log \alpha + \beta \log \kappa(t), \tag{4.10}$$

This model is fitted to the data with linear regression which determines the parameters α and β. For the linear regression to find α and β, all trajectories are low-pass filtered and the beginning and the ending of the trajectories are trimmed based on the first and last sample that reaches half of the median speed. In addition, the parts of the trajectory where the curvature is below its 30 percentile are discarded. These procedures are necessary since curvature-speed power laws are not satisfied at the beginning and end of a movement where the speed is low, neither at low curvature portions where according to (Eq. (4.10)) the speed becomes singular. The procedure examines the duration of segments from the set $\tau_w \in \{0.2, 0.3, ..., 2\}$, measured in seconds.

How well the model approximate the data is computed with the fitness R^2 and is used for further analysis. The R^2 scores are averaged across all sliding windows W_n for each demonstration $demo_i$ and window duration τ_w. This averaged score is:

$$S_{demo_i}(W_n) = \mathbb{E}[R^2].$$

The scores $S_{demo_i}(W_n)$ are averaged across demonstrations only for the segmentation durations for which all scores are larger than a predefined threshold ($R^2 = 0.35$),

$$S(W_n) = \mathbb{E}_i[S_{demo_i}(W_n)].$$

This threshold is used here in order to identify the segment durations that highly comply with the power law. The segment durations for which not all demonstrations resulted in high score are accordingly discarded. Therefore, after this step, only segment durations for which the respective segments comply in average with the power law are used to evaluate the reproductions. The set of scores $\{S(W_n)\}$ is referred to as the "ground-truth" compliance of the task's demonstrations with the power law. The same procedure is carried out to obtain the score of each reproduction, $S_{repro_i}(W_n)$ where only the "ground-truth" segment durations are used for comparison.

For each segment duration, the power law compliance in the reproductions are compared against the power law compliance in the demonstrations as described below. The deviation of each reproduction from the ground-truth score is calculated for each window duration and then averaged over all durations. For each reproduction, a relative score is used which determines the change in the power law fitness from the demonstration's fitness,

$$F(repro_i) = \mathbb{E}_n \left[\frac{S_{repro_i}(W_n) - S(W_n)}{S(W_n)} \right],$$

A threshold $\Delta R^2 = F(repro_i) > 0$ is applied which indicates that the power law competence in the reproduction is as good as the averaged power law competence of the demonstrations.

4.5.4 Implemented model properties

Besides of the motion generation performance delivered by the tested module, it is also interesting to look at the resource management, which can give some insights on the complexity in motion generation methods.

The processing time provides an estimation of the computational complexity of the motion generator algorithm. It corresponds to the amount of time (in millisecond) for the algorithm to provide the next desired state based on the current state of motion.

Category	Property	Measure	Threshold	Scope
Geometric	path	RMSE	-	global
	distance to goal	L2-norm	$1mm$	local
Kinematic	velocity profile	RMSE	–	global
	speed profile	R^2	80%	global
	velocity at goal position	L2-norm	–	local
	movement duration	see Eq. (4.6)	0.1	global
Kinematic & Geometric	trajectory	R^2	95%	global
	power law	ΔR^2	> 0	global
	minimum jerk	RMSD	–	global
Software model	processing time	ms	–	global

Tab. 4.1: Overview of the different categories of measures used in the evaluation of the performance of each participant of the benchmark.

4.6 Statistical analysis of benchmark results

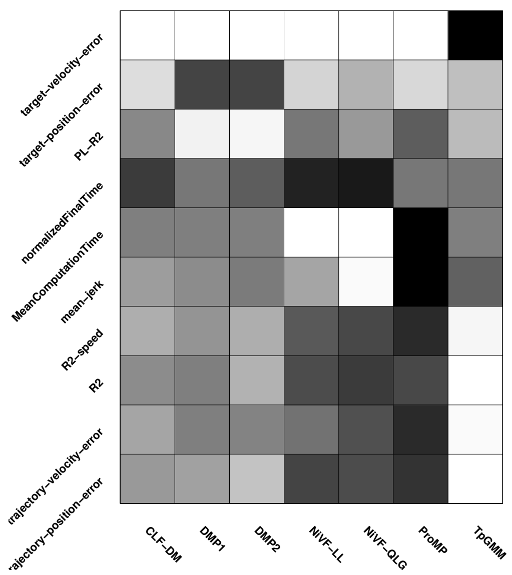
For each benchmark scenario, 100 parameter vectors are drawn from the specified probability distributions, which are described in the previous sections. In Tab. 4.1, an overview of the defined measures is given. Note that all scores whose optimal values may potentially be attained by a good reproduction are equipped with an optimality threshold. For example, all end positions that are in the vicinity of 1mm of the target positions are classified as optimal. However, doing so for the mean squared jerk is artificial and meaningless since an optimal attainable jerk threshold is unknown and only second order time polynomials attain the optimal value of zero mean squared jerk [Polyakov et al., 2009].

4.6.1 Standardized scores and ranking

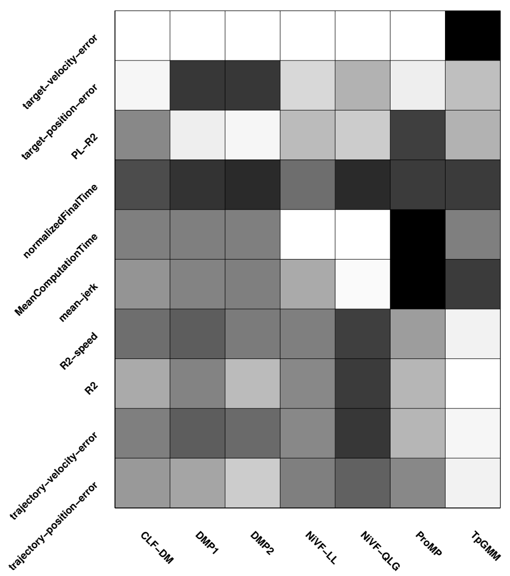
In the benchmark, normalized scores are chosen such that they can be used for the comparison between either among the methods in the benchmark or across shapes in each method. If multiple criteria are used in the performance evaluation the challenge is to avoid that a minority of measures dominate the results. This is tackled by standardized scores to reduce the sensitivity of the scoring system to noise and digitization. In addition a non-parametric method called Kruskal-Wallis test is applied for testing if samples originate from the same distribution is used as a statistical tool to evaluate the performance. Also the parametric equivalent of the Kruskal-Wallis test, the one-way analysis of variance (ANOVA), is available in the benchmark analysis tool. With the help of this tool, each participant can generate standardized plots described to in the following. Fig. 4.4 provides a compact overview of the performance in each measure of every model over all the given benchmark conditions and each condition individually taking also the thresholds into account. For each parameter set, the delivered performance is categorized into high (white), common (gray) and low (black) performance. Each square of this figures summarizes the mean performance over all shapes and over all 100 parameters run in the benchmark. Beginning from this overview one can generate more detailed plots.

4.6.2 Transparency of results

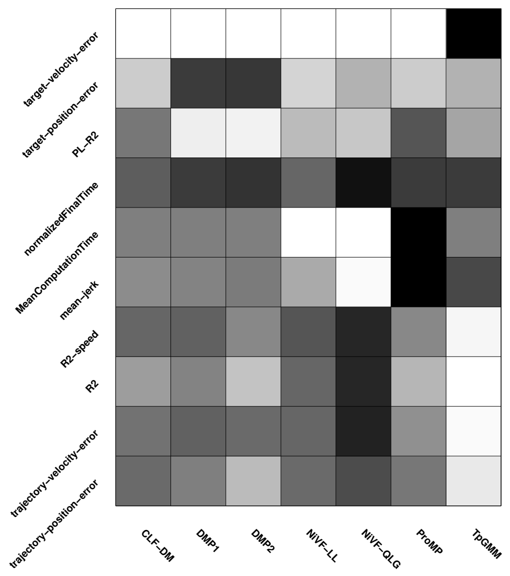
One major requirement for this benchmark is the transparency of the results, which means that all participants should be able to redo all experiments on their own computer to consolidate the results. Therefore, the benchmark software with the benchmark parameters and the first evaluated modules are free for download on: `http://www.amarsi-project.eu/benchmark-framework`.



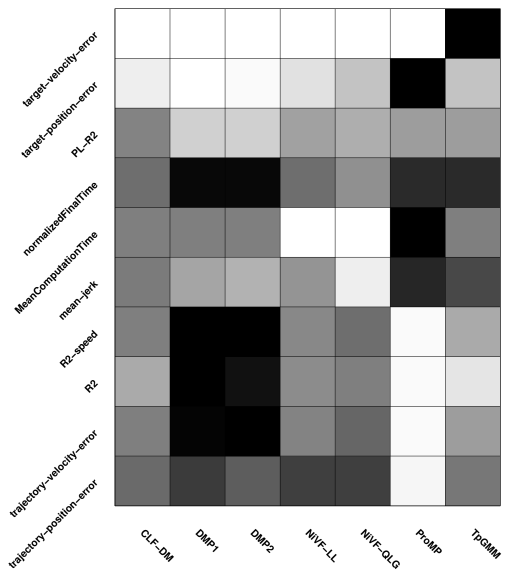
(a) Generalization



(b) Discrete push



(c) Continuous push



(d) Target

Fig. 4.4: General overview of the results from the different benchmark conditions. On the x-axis the names of the different approaches are given. On the y-axis the different measures are listed. Each square can have three different colors indicating either high performance (white), common performance (gray) or low performance (black) in comparison to each other.

4.7 Comparison with focus on NiVF

Five different models are processed for comparison: First a Task-parameterized Gaussian Mixture Model (TpGMM)[Calinon et al., 2013], which implements a virtual spring damper system with variable stiffness and damping. Second, a probabilistic approach called Probabilistic Movement Primitives (ProMP) [Paraschos et al., 2013]. The approach approximates the distribution over the demonstrated trajectories and implements a stochastic feedback controller. Both approaches use internally a second order dynamical system representation, whereas the next approaches are first order dynamical systems. The third approach is called Control Lyapunov Function-based Dynamic Movements (CLF-DM) [Khansari-Zadeh and Billard, 2014]. This approach builds an estimate of an energy function generalized from user demonstrations, which is used during runtime to ensure global asymptotic stability of nonlinear dynamical systems at the target. Furthermore, two variations the Neural imprinted Vector Field (NiVF) (introduced in Chapter 3) are evaluated. The first NiVF version uses a parameterized quadratic Lyapunov candidate L_P and the second version uses a learned Lyapunov candidate L_{ELM} to ensure a stable dynamical system. All models were implemented and provided by the corresponding authors of these models.

Additionally, the dynamic movement primitives approach (DMP) is evaluated. The DMP model is implemented by the author of this thesis and follows the original design of the DMP algorithm described in [Schaal et al., 2005; Ijspeert et al., 2013] and is described in Eq. (2.3). The variation 'DMP2' uses the transformation system proposed in [Park et al., 2008] and is described in Eq. (2.5).

Applied parameters for NiVF learning			
L	Lyapunov candidate	L_P	L_{ELM}
Ω	Workspace	40%	
N_C	# samples drawn from region Ω (Algorithm 3.1)	5000	
ε	Regression parameter	10^{-8}	
R	Number of neurons in the hidden layer	100	
$\mathbf{W}^{inp}$	Initialization range of the input weight matrix (input scaling)	[-1,1]	
$\mathbf{a}$	Initialization of slopes see Eq. (3.5)	1	
$\mathbf{b}$	Initialization ranges of biases see Eq. (3.5)	[-1,1]	

Tab. 4.2: Important parameters which need to be set by the user before network learning. The last set of parameters are needed already for the standard ELM approach. The first part of the table gives the additional parameters important for inducing stability.

4.7.1 Learning and motion generation setup of NiVFs

For each shape in the dataset described in Sec. 4.3 two trained NiVF are provided. The first NiVF denoted as *NiVF_QLG* implements constraints from the parameterized Lyapunov function L_P as described in Sec. 3.4.1 and the second NiVF is denoted as *NiVF_LL* and uses the learned Lyapunov function L_{ELM} as described in Sec. 3.4.2. The applied learning parameters are given in Tab. 4.2. The workspace is 40% bigger that the area covered by training dataset.

The control function used in the benchmark simulation is described in Algorithm 4.3. This function receives the feedback information and incorporates the target movement by correcting the current position accordingly as indicated in Algorithm 4.3 line 1. In case the current position $\mathbf{x}$ is outside of the predefined workspace Ω, the current position is projected to the border $\delta\Omega$ of the workspace. The projected position $\hat{\mathbf{x}}$ is then used to approximate the next velocity command $\hat{\mathbf{v}}$ (compare Algorithm 4.3 line 2-5). The projection to the border of the workspace generalizes the stability constraints to the global space, because on $\delta\Omega$ all velocities are pointing inwards of the workspace given by Eq. (3.12) and inside the Ω the given stability constraints are ensured. The motion generation stops if the produced velocity $\hat{\mathbf{v}}$ is smaller than a tolerance $\varepsilon = 1$. To avoid stopping directly at the start of the motion, this condition is only considered after 40 timesteps.

Algorithm 4.3 Motion generation with NiVF

```
Require: current position $\mathbf{x}(t)$, target position $\mathbf{g}(t)$, accuracy tolerance $\varepsilon$, timestep $t$
Require: trained model $f$
 1: Compute $\hat{\mathbf{x}} \leftarrow \mathbf{x}(t) - \mathbf{g}(t)$
 2: if not $\hat{\mathbf{x}} \in \Omega$ then
 3:     $\hat{\mathbf{x}} \leftarrow$ project $\hat{\mathbf{x}}$ on $\delta\Omega$
 4: end if
 5: Compute $\hat{\mathbf{v}} \leftarrow f(\hat{\mathbf{x}})$ using Eq. (3.2.2)
 6: if $(||\hat{\mathbf{v}}|| < \varepsilon)$ and $(t > 40)$ then
 7:     return $\hat{\mathbf{v}} \leftarrow 0$ and STOP motion
 8: else
 9:     return $\hat{\mathbf{v}}$
10: end if
```

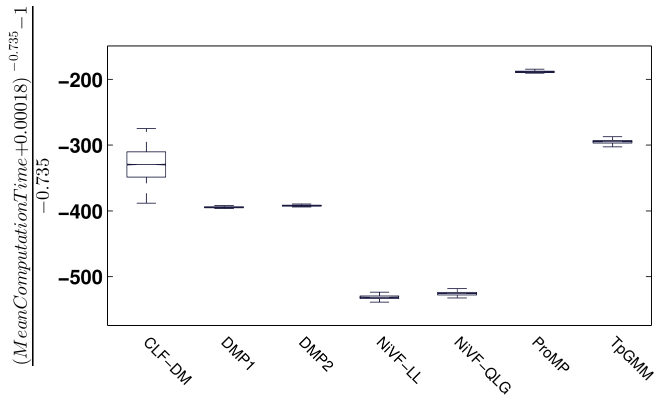


Fig. 4.5: Results of the benchmark comparing five different models over all shapes and benchmark conditions showing the mean computational time during the benchmark tests.

4.7.2 Systematic comparison

The systematic comparison is designed to find comparative results to (i) the robustness of the motion generation (ii) approximation abilities and (iii) human-likeness. The statistical comparisons can be approached from different angels. As aforementioned in Fig. 4.4 the general overview of the results is given and are discussed in more detail.

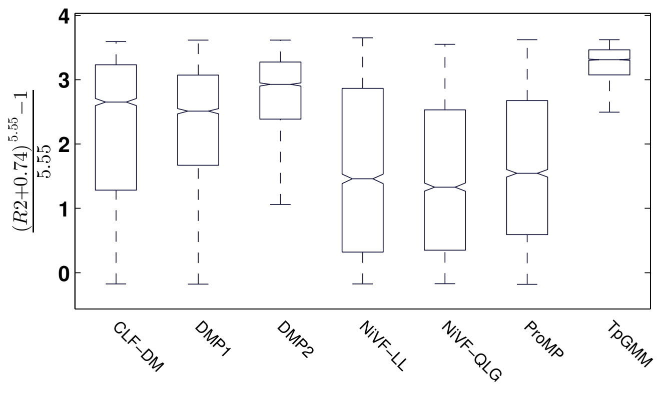
A general result, where the mean computational time is evaluated summarizing all benchmark scenarios over all shapes is shown in Fig. 4.5. The interesting point here is not which model has the fastest computational time, because this is also dependent on the coding of the control function. However, the interesting point is to look at the variances. From all participated modules the 'CLF-DM' has the biggest variance which indicate that the computational effort is depending on the perturbations.

Robustness against perturbation

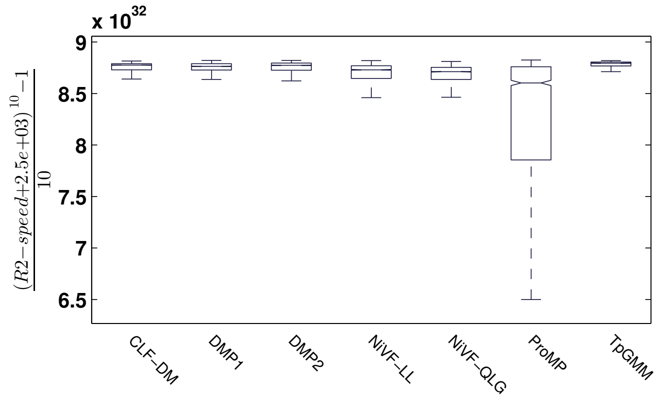
The first question is if the NiVF is robust against perturbation. The overview given in Fig. 4.4 shows in each of the four plots in the first row the trajectory end velocity of the models. All NiVF models have white squares indicating that the optimal zero end velocity is reached (see Sec. 4.5.2), which is only the case if the motion generation has deliberately stopped.

Motion generation accuracy

The ability to generalize the demonstrated path to different initial positions can be analyzed with the 'generalization' scenario of the benchmark shown in Fig. 4.4(a). First the reproduction error of the geo-

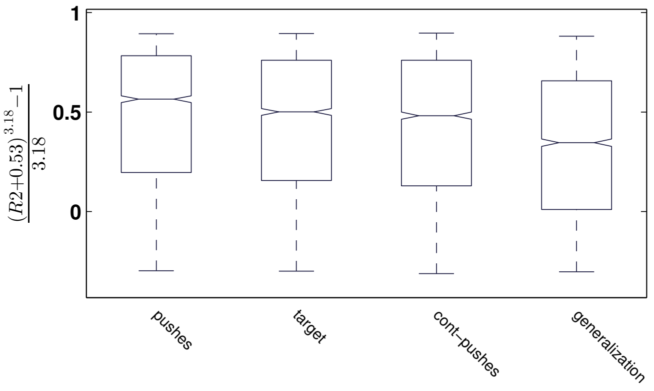


(a) Path R^2

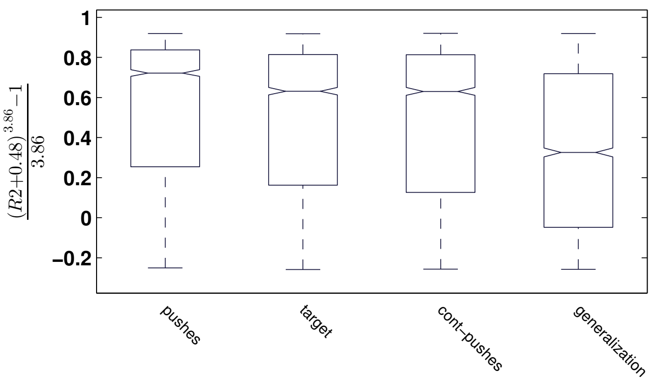


(b) Speed profile R^2

Fig. 4.6: Results of the benchmark comparing five different models over all shapes for the generalization benchmark condition. Fig. 4.6(a) and Fig. 4.6(b) show the shape reproduction performance of the shape and the speed profile respectively according to the R^2 measure.



(a) QLG



(b) LL

Fig. 4.7: Results to the approximation performance over all benchmark scenarios of the NiVF for both Lyapunov candidates: the quadratic Lyapunov candidate in Fig. 4.7(a) and the learned Lyapunov candidate in Fig. 4.7(b).

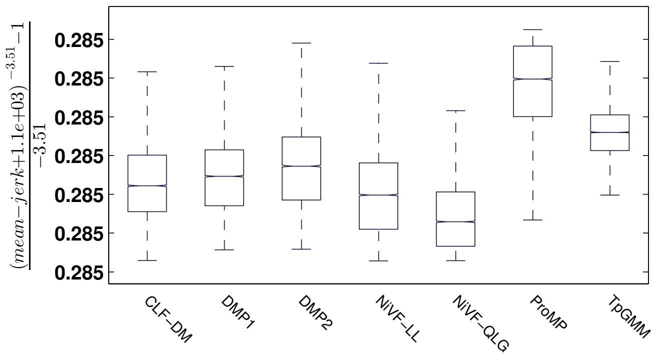


Fig. 4.8: Results of the benchmark comparing five different models over all shapes and benchmark conditions showing the mean jerk of all produced trajectories of each model.

metric shape measured with R^2 on the path information and also the approximation ability to reproduce the speed profile for all participating models is shown in Fig. 4.6. In Fig. 4.6(a) the results indicate that the 'DMP2' and the 'TpGMM' models are particularly well in approximating the shape in terms of different initial start positions. The 'DMP2' is invariant to the start and goal position as discussed in Sec. 2.2.3. Also the 'TpGMM' model is parameterized with time and spatial information to reproduce the learned demonstrations. Both models are also non-autonomous dynamical systems. On the other side there are the autonomous dynamical systems given by the NiVF versions and the 'CLF-DM'. The latter uses an online correction term to react to the perturbations such that the trajectory generation returns to the learned trajectory. In Fig. 4.6(b), a surprising result is that the 'ProbDMP' does change the speed profile drastically compared to the demonstrations. All other methods are approximating the speed profile more accurate.

In Fig. 4.7, the comparison of the reproduction error according to the trajectory R^2 measure is shown between the four different benchmark scenarios for the NiVF with constraints from the quadratic Lyapunov candidate (Fig. 4.7(a)) and with the learned Lyapunov function in Fig. 4.7(b). One can see in both plots that they perform worst in the perturbation type 'generalization'. The perturbation types 'target' and 'cont-push' have similar performance, which is plaussible because the target change is used to perturb the current position (see Algorithm 4.3 line 1) and becomes a continuous perturbation to the end-effector.

As discussed in Chapter 3 the parameterized quadratic Lyapunov candidate (NiVF_QLG) allows less complex dynamics than the Lyapunov candidate learned by the ELM (NiVF_LL). Compared to the provided evaluation in Chapter 3 the starting points are more diverse and thus are further away from the training data. Depending on the shape, the vector field cannot generalize this motion to arbitrary starting points, but it does generate a smooth trajectory to the target point. In addition, depending on the size of the workspace, some of the requested starting points can be outside of the workspace. Both NiVF variants have the same stability constraints on the border of the workspace which can be the reason why the performance in the generalization scenario are similar. Because if the start point is outside of the workspace a smooth trajectory is generated to the predefined workspace and then the movement is proceeded depending on the shape of the learned vector field.

Human-likeness

To evaluate the human-likeness of all models, two different analysis are applied. First, the general features of smoothness are evaluated by the jerk measure (compare Fig. 4.8) and second the comparisons against the original demonstrated training data with the power law measure. In the comparison all models of the participants generate similar smooth motions.

The speed profile of each tested method is shown in Fig. 4.6(b). The compliance of the reproduced trajectories to the power law is selected to evaluate the human-likeness. In Fig. 4.9, the results for each

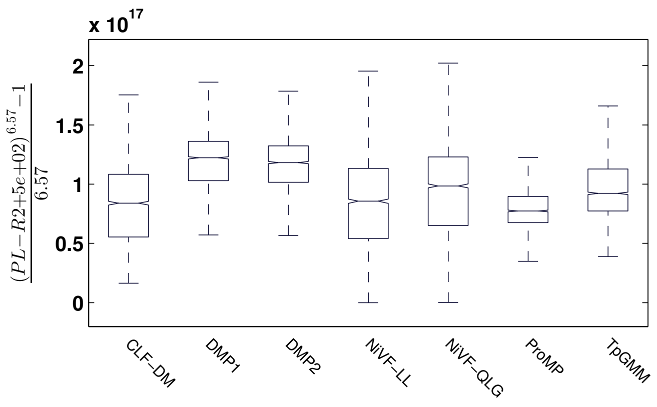


Fig. 4.9: Results of the benchmark scenario 'generalization' comparing all models regarding the approximation ability of the relation of curvature and speed from the demonstration (power law Eq. (4.10)).

shape for the generalization benchmark scenarios are shown.

All methods have scores bigger that zero which means that the delivered performance of the methods is even more compliance to the power law than the human demonstrations. Regarding the NiVF models one can see that the lower percentiles are close to zero which indicate a similar level of complaints to the power law as evaluated in the human demonstrations. However, due to the fact that the models are not invariant to start and goal positions it is plausible that the higher percentiles are far away from zero, because although the trajectories are complaint to the power law, they do not show the same characteristics as the human demonstrations.

Summary

The analysis shows that (i) the module is robust to perturbations and reliably reaches the target. Regarding the (ii) approximation abilities it became clear that the demonstrated path was not always reproduced. The reason for this behavior is the fact that the NiVF is not invariant to the start and goal state, which can be easily approached by adding an affine transformation to compensate for the change in the start goal relation. However, motion shapes are actually smoothly adopted across the workspace. At last, (iii) the human-likeness of the generated trajectory is indicated by a smooth motion generation and a good compliance to the movement power law.

4.8 Conclusion

This benchmark framework was developed to compare state-of-the-art movement primitive representations that cover human reaching motions. It offers to the community standardized and systematic comparisons for the increasing variety of the different approaches. This benchmark evaluates these methods on a dataset of human motions based on criteria such as 'level of similarity to human motions', 'accuracy in reaching the goal state', 'adaptability to changes in dynamic environments', 'robustness to perturbations', etc.

The contribution of this work is threefold. First, four different scenarios for motion generation in a 2D drawing task are proposed in which specific and controlled perturbations are simulated during the motion generation. Second, a set of different scores evaluating specific features of the motion are suggested. The specific features are chosen to evaluate kinematic, geometric and a mixture of both properties. By dividing the evaluation into these different categorical properties, one can reveal weaknesses and strengths of different movement primitive representations. At last, first comparisons of state-of-the-art movement primitive representations with focus on the NiVF are provided.

CHAPTER 5

ADAPTIVE SEQUENCING OF MOVEMENT PRIMITIVES WITH NEURAL NETWORKS

In the previous chapters, the representations and learning methods are focused on only one single movement primitive. The important properties are identified which specify what features need to be implemented in these movement primitive representations for robotic motion generation.

In this chapter, a movement primitive library for sequencing stored movement primitives to compose complex motions is considered. The movement primitive library is designed as one single neural network with the functionality to learn multiple movement primitives and also the activation sequence of the learned primitives for complex motions. This new approach is not limited to sequencing movement primitives but can be used also for parallel blending of movement primitives. Results show that complex motions can be composed from single motion primitives efficiently in one single feed-forward neural network.

5.1 Generation of complex movements from movement primitives

In the recent years, the creation of motion primitive libraries became considerably important in robotics. The issue of sequencing movement primitives to compose complex trajectories is usually discussed separately from the problems of how to represent movement primitives and learn generic motion skill in a full architecture.

Learning of movements consists in storing the sequential activation of each specific movement primitive [Kulic et al., 2008a]. Rhodes et. al. [Rhodes et al., 2004] discussed the possibility of short sequences organized as task specific chunks, which can be activated during cognitive operations to allow rapid processing. In [Luksch et al., 2012a,b] continuous-time recurrent neural networks (CTRNNs) are used to model conditioned selections of primitives dependent on sensory data. This control scheme uses activation patterns which describe either the execution of primitives one after the other or with continuous blending. This approach can be used in addition to the proposed monolithic architecture to include a reactive behavior, if perturbations occur. This becomes possible because each neuron of the CTRNN represents one of the movement primitives controlled by the activation dynamics. In [Murata et al., 2013], a stochastic continuous-time recurrent neural network (S-CTRNN) is introduced. The S-CTRNN is able to learn and reproduce trajectories by extracting the stochastic structures hidden in the demonstrated training trajectories. One advantage of this method is that learning and smooth blending of multiple time series is combined in one single network. However, the parameterization of the learned time series are encoded in so called context parameters, which are generated in an unsupervised fashion. These parameters are not constant for one time series, which makes it difficult to use this method in a larger architecture.

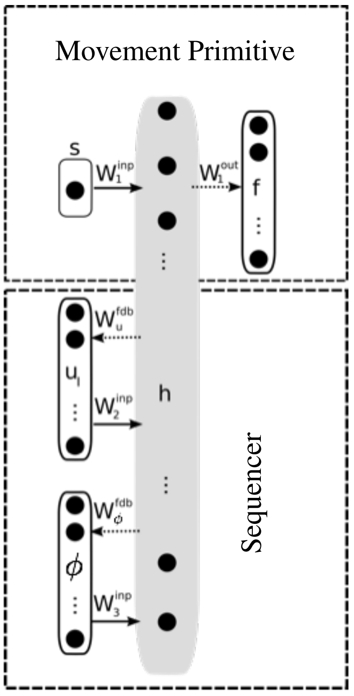


Fig. 5.1: This figure illustrates a holistic neural network architectures, which represent a single network solution that is able to learn a sequence of starting points $\mathbf{u}_l$ and addresses ϕ together with a motion primitive representation. The addresses coded in ϕ are used to select a respective motion pattern.

In all approaches based on dynamical systems, a minimum control logic for sequencing motion primitives has to (i) index the primitives to be able to address them in a larger architecture and (ii) parametrize the execution by the starting point and the goal. Encoding of sequences of motion primitives requires to store the sequence of starting points (in absolute or relative coordinates) together with an identification specifying the learned motions in the library. It could e.g. be realized either with brute force by memorizing respective data or by learning a respective sequence with any method for sequence learning (see [Amit and Matari, 2002; Kulic et al., 2008a]). In this thesis an architecture based on a neural network is proposed to represent both the motion primitives and the temporal sequence of primitives.

In this chapter, the general assumption is that there can be multiple primitives starting at the same point relative to the goal, but following different motion patterns. The ELM-based neural network with output feedback (described in Sec. 5.2) is used to store movement primitives and their sequence of activation in a single shared representation. Additionally, this shared representation is exploited to blend two learned movement primitives.

5.2 Holistic neural learning architecture

This section describes the neural network architecture that stores movement primitives together with the activation sequence. The idea to generate complex movements from movement primitives with one neural network architecture requires to solve two tasks: (i) the motion primitive representation itself and (ii) to store the activation sequence of each movement primitive together with all necessary parameters. In Fig. 5.1, a possible combination of motion primitive representation and sequencing is shown. These mappings, which are usually considered separately, operate here on a shared hidden representation **h**.

5.2.1 Representation of movement primitives

In Chapter 3 and Chapter 4, different representations of movement primitives are discussed by focusing on the requirement for stability in the underlying dynamical systems. One example of data-driven learning of dynamical systems with stability constraints is discussed in Chapter 3. In this chapter, the movement primitives are considered to be Dynamic Movement Primitives (DMPs) (compare Sec. 2.2.3) as described in the following.

Spring damper system of DMPs

The DMP approach uses a transformation system to address the stability concerns

$$\tau \ddot{u}_t = K(g - u_t) - D\dot{u}_t - Ks(g - u_0) + Kf(s), \tag{5.1}$$

coupled with a canonical system:

$$\dot{s}_t = -\frac{1}{M_{\mathrm{mp}}}, \tag{5.2}$$

where K, D are stiffness and damping constants with $D = 2\sqrt{K}$ to generate a critical damped system [Park et al., 2008]. A linear canonical system is used and after the specified number of time steps M_{mp} the motion generation will stop. The stability of this dynamical system is ensured in case that the perturbation f becomes zero at the end of the movement, which results in a linear convergence to the goal point. The transformation system given in Eq. (5.1) is a variation of the original formulation of the DMP approach proposed in [Ijspeert et al., 2003]. The major difference is that the transformation system described in Eq. (5.1) is invariant to the relative position of the start and goal position. In [Park et al., 2008], the properties of this transformation system are described in more detail.

The transformation system is a linear dynamical system which is perturbed by a non linearity function f to represent arbitrarily shaped and smooth reaching motions. Typically, the function f can be approximated by a set of Gaussian basis functions. In the following the proposed DMP model uses an ELM to learn f. The data to train the ELM is given by rewriting Eq. (5.1) such as:

$$f(s) = -(g - u_t) + \frac{D}{K}\dot{u}_t + s(g - u_0) + \frac{1}{K}\ddot{u}_t. \tag{5.3}$$

The generation of movement primitives is performed in a normalized space, where the goal point is always the origin. This allows to shift the movement to any location by adding an offset to the whole generated trajectory.

Neural network dynamics

In a single neural architecture it is also beneficial to use this normalized space because then learning requires only the initial start point relative to the goal point of the demonstrated trajectories as additional parameters for each movement primitive to reproduce the demonstration. In the proposed architecture illustrated in Fig. 5.1, the function $f(s)$ is approximated by the upper part of the architecture.

The representation of movement primitives can be modeled by ELM architectures (see Fig. 5.1 top section). The following ELM network dynamics are considered for the proposed learning architecture, if a movement is generated:

$$\mathbf{h}(k) = \sigma(W_1^{\mathbf{inp}} s(k) + W_2^{\mathbf{inp}} \mathbf{u}_l + \mathbf{W}_3^{inp} \phi + b), \tag{5.4}$$

$$\hat{\mathbf{f}}(k) = W_1^{\mathbf{out}} \mathbf{H}(k), \tag{5.5}$$

where $\mathbf{h} \in \mathbb{R}^R$ gives the hidden state and $\phi \in \mathbb{R}^I$, $\mathbf{u}_l \in \mathbb{R}^d$ denote the constant input of the network. These constant inputs are respectively connected to the hidden layer through the input matrices $W_3^{\mathbf{inp}} \in \mathbb{R}^{R \times I}$ and $\mathbf{W}_2^{inp} \in \mathbb{R}^{R \times d}$, where I is the expected number of movement primitives stored in this network and d is the dimension of the normalized movement primitive space. The bias of the hidden layer neurons is denoted by b and both input weight matrices remain fixed after random initialization. The activation function $\sigma(x) = 1/(1 + \exp(-x))$ is applied to each neuron in the hidden layer. The hidden layer is connected to the output through the output matrix $W_1^{\mathbf{out}} \in \mathbb{R}^{d \times R}$ and gives the approximation $\hat{\mathbf{f}}$ of the perturbation of the transformation system given in Eq. (5.1). At the beginning of the movement generation the input is $s = 1$ and decreases to $s = 0$ which corresponds to the end of the motion (see Eq. (5.2)).

5.2.2 Organization of movement primitives in a sequence

The task of the sequencer (bottom part in Fig. 5.1) is to activate a number of movement primitives in a coordinated way. Two conditions need to be met in order to implement such behavior.

First a movement primitive needs to be found in the library, which requires an address that uniquely identifies the chosen movement primitive. Note that the address does not necessarily need to have any semantics. The address is an identifier and could even be chosen randomly, however, in the following a one-of-K-coding vector ϕ is chosen. Second, the initial conditions of the movement primitive need to set e.g. start and goal point. The considered normalized movement primitive space forces the goal point to be constant at the origin. Therefore, only the start point $\mathbf{u}_l$ in the normalized space is needed for the initial condition of the chosen movement primitive.

The representation of a sequence of movement primitives can be modeled by ELM architectures with output feedback (see the sequencer in Fig. 5.1 bottom section). The ELM has two inputs to incorporate the information coding the sequence of the identification and start parameters of the movement primitives participating a complex movement. The following ELM network dynamics are considered, if a sequence is recalled:

$$\mathbf{h}(k+1) = \sigma(\mathbf{W}_2^{inp}\mathbf{u}_l(k) + \mathbf{W}_3^{inp}\phi(k) + \mathbf{b}), \tag{5.6}$$

$$\phi(k+1) = \mathbf{W}_\phi^{fdb}\mathbf{h}(k), \tag{5.7}$$

$$\mathbf{u}_l(k+1) = \mathbf{W}_u^{fdb}\mathbf{h}(k), \tag{5.8}$$

where $\hat{f}$ is equal to Eq. (5.5). The hidden state is given by $\mathbf{h} \in \mathbb{R}^R$ and $\phi \in \mathbb{R}^I$, $\mathbf{u}_l \in \mathbb{R}^d$ denote again the inputs of the network. The inputs are connected to the hidden layer through the input matrices $\mathbf{W}_3^{inp} \in \mathbb{R}^{R\times I}$ and $\mathbf{W}_2^{inp} \in \mathbb{R}^{R\times d}$, where I is the expected number of movement primitives stored in this network and d is the dimension of the normalized movement primitive space. The phase variable s is connected to the hidden layer by $\mathbf{W}_1^{inp} \in \mathbb{R}^{R\times 1}$. The bias of the hidden layer neurons is denoted by b and both input weight matrices remain fixed after random initialization. The activation function $\sigma(x) = 1/(1+\exp(-x))$ is applied to each neuron in the hidden layer. The hidden layer is fed back to the inputs through matrices $\mathbf{W}_\phi^{fdb} \in \mathbb{R}^{I\times R}$ and $\mathbf{W}_u^{fdb} \in \mathbb{R}^{d\times R}$.

Note that these dynamics are activated only if the input of the upper part of the proposed architecture is $s = 0$. During the movement generation i.e. $s > 0$ the input of the lower part stays fixed. Fixing the inputs effectively cuts of the recurrence in the lower part, which means Eq. (5.7) and Eq. (5.8) are not applied. This parametrizes the particular movement primitive, because of its unique impact on the hidden layer representation.

5.3 Learning methodology

The movement primitives and the sequencer are learned in a supervised manner. In the following, the training data for each learning task is introduced.

5.3.1 Training data

Before the learning methodology is introduced, the training data is specified: (i) for the movement primitive representation and (ii) the sequence of activations.

Movement primitive

The training data for learning movement primitives is specified by the address ϕ and at least one demonstration. Each demonstration is normalized before learning such that the goal point is set to the origin of the normalized space. The input s is given by a decreasing linear function from $s = 1$ to $s = 0$ in M_{mp} steps, where M_{mp} is the amount of samples in the demonstrated trajectory. Again the ϕ and starting point of the normalized training trajectory $\mathbf{u}_l$ are given and the input $(\phi, \mathbf{u}_l)$ stays fixed during the learning of the DMP. The hidden layer states are obtained from S and collected in the matrix $\mathbf{H}(S(j)) = (\mathbf{h}(s(1)), \ldots, \mathbf{h}(s(M_{\mathrm{mp}})))$, where $j \in [1 \ldots M_{\mathrm{mp}}]$ is the current time-step. The corresponding targets are given by $\mathbf{T}(j) = (f(s(1)), \ldots, f(s(M_{\mathrm{mp}})))$, where f is given by Eq. (5.3).

Sequence of addresses

The sequence of parametrization $(\phi, \mathbf{u}_l)$ is learned by closing the loop (e.i. Eq. (5.7) and Eq. (5.8) are applied) after training to replay the respective sequence of starting points and addresses. In a standard approach it leads to a hierarchical setup, where the (neural) sequencer provides the parametrization. An additional control scheme is required to retrieve the respective primitive and execute the movement until it is converged to the (intermediate) goal point.

The input at iteration k in the sequence is given by $\mathbf{x}(k) = (\mathbf{u}_l(k), \phi(k))$ and the corresponding target $\mathbf{t}(k) = (\mathbf{u}_l(k+1), \phi(k+1))$. If M_{tr} gives the length of the sequence training data then a sequence is given by $\mathbf{S} = (\mathbf{X}, \mathbf{T}) = (\mathbf{x}(n), \mathbf{t}(n)) : n = 1 \ldots M_{\mathrm{tr}}$.

5.3.2 Efficient online learning

In this section, supervised learning schemes for the read-out layers are introduced. In principle any supervised learning rule could be applied. For the sake of simplicity, only the two learning rules utilized in this thesis are discussed comprising a standard offline learning scheme based on regularized linear regression and an efficient online learning rule. The training data is considered to be organized as follows: the matrix $\mathbf{H}(\mathbf{X}) = (\mathbf{h}(\mathbf{x}(1)), \ldots, \mathbf{h}(\mathbf{x}(N_{\mathrm{tr}})))$ collects the hidden layer states obtained from a given dataset $\mathbf{D} = (\mathbf{X}, \mathbf{T}) = (\mathbf{x}(n), \mathbf{t}(n)) : n = 1 \ldots N_{\mathrm{tr}}$ for inputs $\mathbf{X}$ and the corresponding output targets $\mathbf{T}$. The task is to minimize the error between the targets T and the estimations from the learning network $(\mathbf{W} \cdot \mathbf{H}(\mathbf{X}))$ with respect of the read-out weights $\mathbf{W}$.

$$\mathbf{W}^{out} = \underset{\mathbf{W}}{\arg\min}(\|\mathbf{W} \cdot \mathbf{H}(\mathbf{X}) - \mathbf{T}\|^2 + \varepsilon \|\mathbf{W}\|^2) \tag{5.9}$$

where large weights are penalized by $\varepsilon \|\mathbf{W}\|^2$. The factor $\varepsilon > 0$ is used to regularize the read-out weights $\mathbf{W}^{out}$. In the following two possible solutions to the minimization problem Eq. (5.9) are introduced.

The ELM training uses learning schema of online sequential ELM (OS-ELM [Liang et al., 2006]). This allows to additionally learn primitives or refine the movement primitives if new training data is available. The initial learning phase and a sequential learning phase are explained in the following.

1. The first primitive is used in the initial learning phase with $k = 0$:

$$\mathbf{W}_k^{out} = \mathbf{P}_k \mathbf{H}_k^T \mathbf{T}_k, \tag{5.10}$$

$$\mathbf{P}_k = (\mathbf{H}_k^T \mathbf{H}_k + \varepsilon \mathbb{1})^{-1}, \tag{5.11}$$

   where ε is again the regression parameter and $\mathbb{1}$ is the identity matrix.

2. Each additional primitives can be added in the sequential learning phase:

$$\mathbf{P}_{k+1} = \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbb{1} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k, \tag{5.12}$$

$$\mathbf{W}_{k+1}^{out} = \mathbf{W}_k^{out} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \mathbf{T}_k). \tag{5.13}$$

Note that, for the initial phase, the first demonstration hat to include at least so many training samples as the number of hidden neurons in the ELM.

5.4 Composition of complex trajectories

In this section, a qualitative evaluation of the neural network is described that combines sequencing and movement primitive representation. For each motion pattern in the LASA dataset one primitive is learned and is stored in the network.

5.4.1 Recall of an activation sequence with start parameters

In this section, training of the sequencing part (bottom part in Fig. 5.1) is evaluated. The task of the sequencer is not only to sequence the ϕs in the right order but also to give the parameterization, which is in this context the starting point $\mathbf{u}_l$ of the motion relative to the end point. An update of the sequencing part according to Eq. (5.6) generates a new starting point to parametrize a new primitive and an index to select the respective primitive. In any case, the input is $s = 0$ and therefore the sequencing part can be trained independently of the movement primitive part. An example of a reproduced learned initial point sequence is given in Fig. 5.2. Here we used $\varepsilon_{\text{Seq}} = 1$ for adapting the feedback weights $\mathbf{W}_*^{fdb}$. Note that no preprocessing of the starting points (see Fig. 5.2(circles)) is necessary for learning. After learning the network generates a mean starting point (see Fig. 5.2(cross)) corresponding to the primitive ϕs (indicated as numbers in Fig. 5.2).

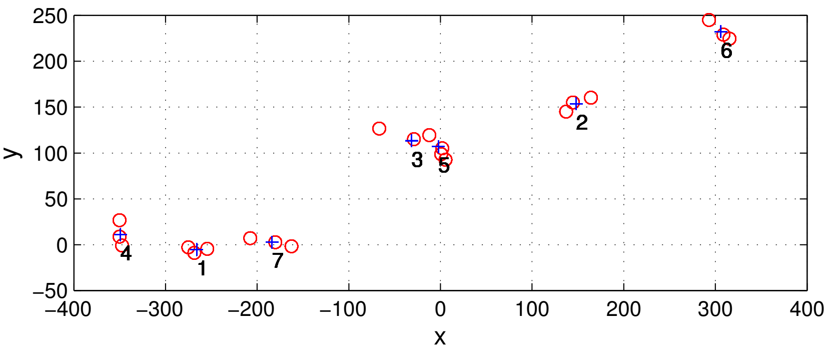


Fig. 5.2: Reproduction of the learned sequence of starting points (in mm). Red circles mark the starting points in the training data and the blue cross marks the reproduced starting point associated with the ϕ noted below the points. Note that primitive three and five almost share a common starting point relatively to the goal point.

5.4.2 Memory capacity and reproduction accuracy

In this experiment, the combination of a sequencer and the DMP approach to represent motions in this monolithic approach (Fig. 5.1) is considered.

After learning, the memory capacity of the network is evaluated, where the reproduction performance of the motion generation depending on the size of the hidden layer is measured. In total $M = 20$ motion patterns are learned, where each motion pattern is specified by N demonstrations. The order in which the motion patterns are presented is randomized. The reproduction performance of the monolithic approach over $k = 10$ network initialization for the hidden layer size of $h_{\text{dim}} \in \{40, \dots, 200\}$ is evaluated. The input matrices and biases b_i are initialized randomly from uniform distributions in $[-10, 10]$ and $[-1, 1]$ respectively. The DMP transformation system is initialized with $K = 200$ (see Eq. (5.1)). The motion starts with the initialized canonical system where $s = 1$ and stops if the phase variable $s < 0$. The reproduction error is measured by the point wise root mean square error (RMSE) over all motion patterns learned by the network:

$$RMSE = \frac{1}{N} \sum_{N} \sqrt{\frac{1}{N_{tr}} \sum_{i}^{N_{tr}} ||u_n(i) - \hat{u}_n(i)||^2},$$

where u is the position at time-step i and $\hat{u}$ is the corresponding reproduction. For this experiment the ability to store a number of movement primitives is considered, therefore no sequences are implemented but the ϕ address of the desired primitive is set manually in the input layer, i.e. feedback is cut off, thus no update of the input $(\phi, \mathbf{u}_l)$ is readout by $\mathbf{W}_*^{fdb}$.

In Fig. 5.3 the *RMSE* is illustrated. Note that the performance is robust over the number of neurons in the hidden layer. These results indicate that for learning 20 motion patterns, the hidden layer performs best with at least 140 or more neurons.

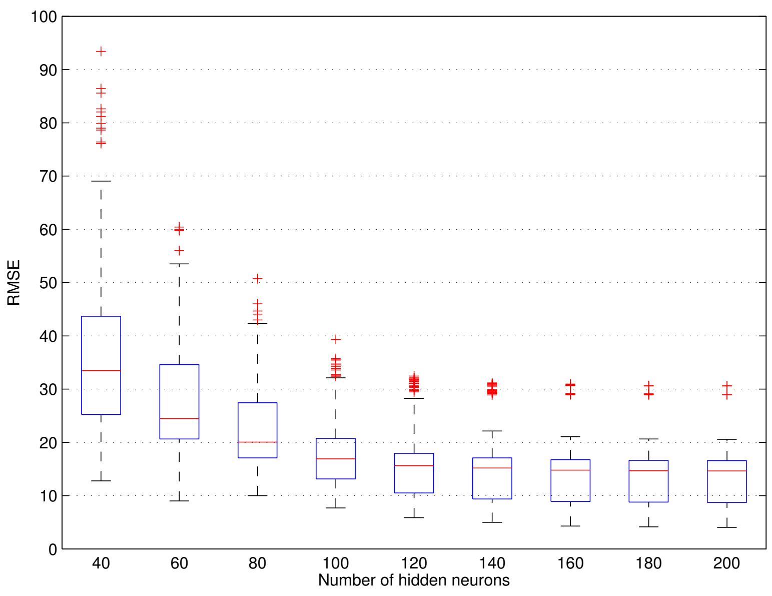


Fig. 5.3: Learning of $M = 20$ motion patterns in ELMs using different hidden layer size. The x-axis show how many neurons are used in the hidden layer.

5.4.3 Combined sequencing and motion generation

After training seven primitives and an activation sequence of these primitives, the network can be exploited with a minimal overhead of additional "control". Only two simple operations have to be performed. While executing the movement primitive, the feedback from the network to the sequencing part is inhibited i.e. Eq. (5.7) and Eq. (5.8) are not applied, until the movement primitive is finished. After the motion generation (i.e. $s = 0$) the Eq. (5.7) and Eq. (5.8) are applied and a new ϕ and u_l can be recalled from the network. The inhibition could be implemented neurally by means of a gating neuron which is excited by the motion primitive input and allows for an update step of the sequencer only if the state s of the motion primitive input reaches zero. Then, an update of the lower sequencing loop is performed and the newly created offset position $\mathbf{u}_l$ needs to be copied to the state of the primitive. Finally the next motion primitive can start, as selected through the newly updated index. In Fig. 5.4, a complex trajectory is shown which is composed of seven movement primitives generated by the proposed method. The complex movement is repeated three times with the three different initial positions (see Fig. 5.4 zoomed area upper right corner). The overall sequence structure repeats itself, because a cyclic loop was trained for the sequencer. It is interesting to note that after the first primitive ('sine wave'), the next primitive ('Sshape') all three repetitions of the complex motions are aligned (see Fig. 5.4 zoomed area bottom left corner). This indicates that the sequencer has learned the parameterization very robustly and can handle noise in the initial $\mathbf{u}_l$. A pseudo code of the complex motion generation with stored movement primitives is provided in Algorithm 5.4. The movement primitive generation is given in lines 8-12 and the sequencer code is wrapped around in lines 4-7 and lines 13-15.

5.4.4 Blending of two movement primitives

Due to the one-of-K coding scheme only one neuron in the identification vector ϕ is active represent each learned movement primitive. In this experiment two motion primitives are triggered in parallel.

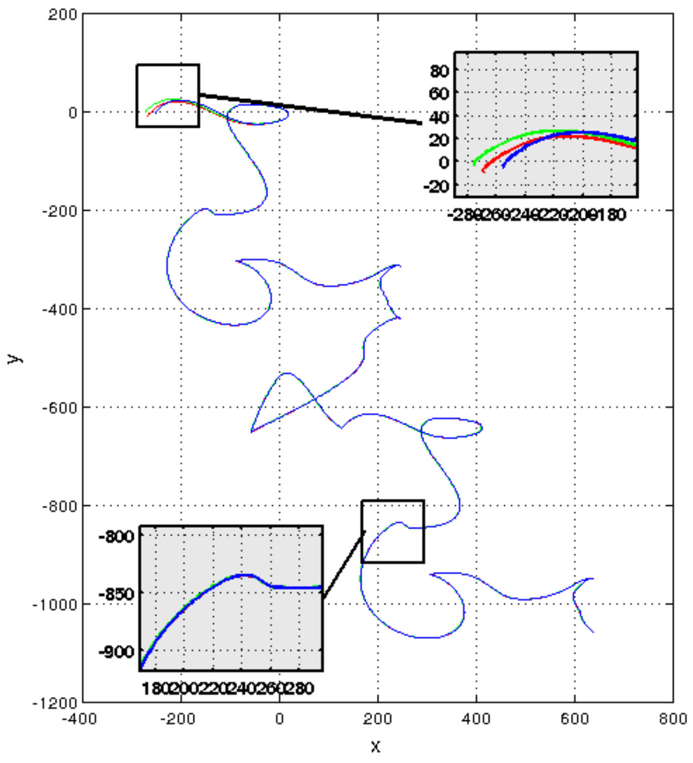


Fig. 5.4: Complex trajectory composed of seven sequenced movement primitives. The sequence is repeated twice. Note that the complete sequence is started from three different starting points (see zoomed display at the beginning of the complex motion).

Algorithm 5.4 Execute complex motions

```
Require: initial start point $\mathbf{u}(0)$ and first movement primitive given by $\phi(0)$
 1: set $k = 0$
 2: set ELM input $s = 0$, $\mathbf{u}_l = \mathbf{u}(k)$, $\phi = \phi(k)$
 3: update hidden state Eq. (5.6)
 4: repeat
 5:     set ELM input $s = 1$
 6:     update hidden state Eq. (5.6)
 7:     set DMP start point $\mathbf{u}_0 = \mathbf{u}_l$ and goal point $g = 0$
 8:     repeat
 9:         read out $\hat{f}(s)$ following Eq. (5.5)
10:         update DMP transformation system Eq. (5.1) with $\hat{f}$
11:         update input $s$ with DMP canonical system Eq. (5.2)
12:     until $s < 0$
13:     set ELM input $s = 0$
14:     update hidden state Eq. (5.6)
15:     read out $\phi(k+1)$ and $\mathbf{u}_l(k+1)$ following Eq. (5.7) and Eq. (5.8)
16:     $k \leftarrow k + 1$
17: until Complex motion is complete
```

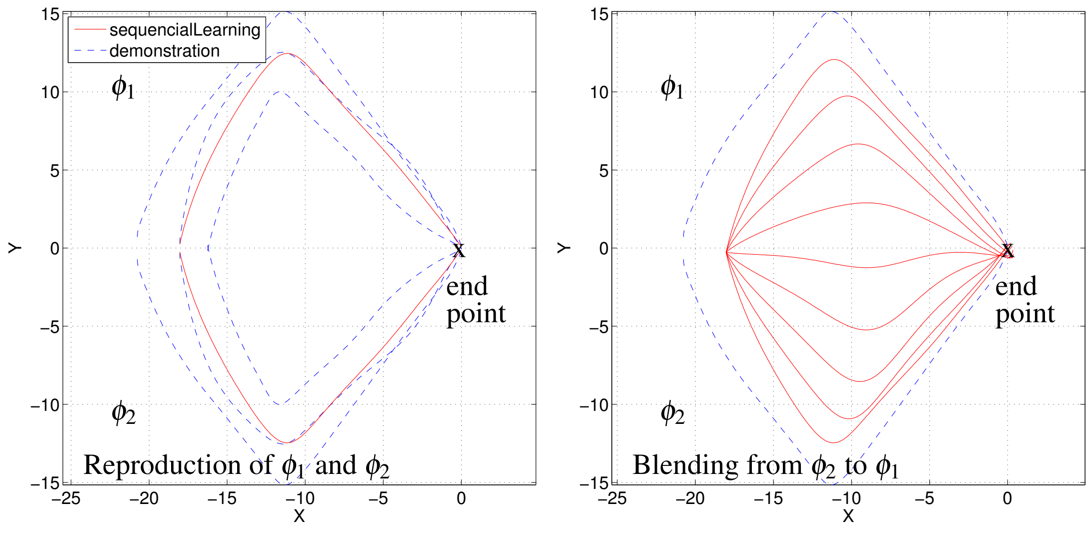


Fig. 5.5: Two motions are learned in one network. One is the original 'Angle' shape from the LASA dataset and the other is the mirrored version of this motion pattern. The two learned motions (left) and the smooth blending (right). We show the training data (blue) and the reproduction (red).

The assumption is that if two neurons get activated, the global representation creates a mixture of the original learned motions, which is called blending.

The input is given by the phase variable s, which is initially set to one, and the ϕ vector of the chosen primitive ($\phi_1 = (1,0)$ and $\phi_2 = (0,1)$). The parameter $\varepsilon_{\mathrm{MP}} = 10^{-5}$ is used for leaning the motion shape (i.e. $\mathbf{W}_1^{out}$) and $\varepsilon_{\mathrm{Seq}} = 1$ is used for learning the sequencer (i.e. $\mathbf{W}_*^{fdb}$). Note that the regression parameters are very differently due to the complexity of the different tasks. The movement primitive representation requires more model complexity, thus a small ε is used. All other variables are initialized as described in the previous section. Seven new ϕs are generated linearly spaced between ϕ_1 and ϕ_2. The 'Angle' shape (ϕ_1) from the LASA dataset is used and the same shape mirrored with respect to the x-axis (ϕ_2) is used as training data to show the blending behavior.

The demonstrated (blue dashed line) and reproduced movements (red) can be seen in Fig. 5.5(left), if solely ϕ_1 or ϕ_2 are fed to the network's input. The blended movements are shown in Fig. 5.5(right). Note that the movement primitives ϕ_1 and ϕ_2 are reproduced accurately and that a smooth transition from ϕ_1 or ϕ_2 is possible in this architecture simply by changing the input linear from ϕ_1 to ϕ_2.

5.5 Discussion and perspectives

In this chapter a monolithic neural network approach is proposed in which a number of different movement primitives can effectively be learned within a single shared representation and can be sequenced by applying the respective input for parameterization.

The number of movement primitives one wants to store in the architecture has to be known a priory due to the chosen one-of-K encoding, because only one neuron is active per learned movement primitive. All other neurons in the one-of-K encoding are zero and do not contribute to the hidden representation. In order to solve this problem the idea is to enlarge dynamically the ϕ vector as new movement primitives are required to be stored.

Although a shared hidden representation is used, one can decouple the learning of the motion primitives and the sequencing part. This is possible because of the assumption that the sequencer is only activated if the last motion primitive has finished i.e. the input s is always zero. One could argue that, technically, this shown combination of learned generic sequencing of movement primitives is providing a semantic hierarchy: the 'upper level' sequencer controls the 'lower level' movement primitives. The implementation, however, uses a single shared representation and training scheme for both skills. Note that this shared representation is beneficial for blending two motions, which is not a trivial matter if sequencing and movement primitives operate separately.

In future work the used non-recurrent hidden representations in the neural network could be expanded to a recurrent hidden representation. These are reasonable options: if the presentation of training data is ordered along the demonstrated trajectories, possibly online, then the temporal memory provided by a reservoir is useful and helps learning as e.g. shown in earlier work (e.g.[Rolf et al., 2009]); if data is randomized, which is also feasible because the underlying mapping to be learned is static in nature, then recurrence in the hidden layer is not essential and a simpler model that is not recurrent in the hidden layer can be used. Such a model has been very recently introduced in the context of learning inverse kinematics and attractor based motion generation in [Reinhart and Steil, 2011] and is discussed in depth in [Reinhart, 2012].

5.6 Conclusion

A single movement primitive is understood to be a small entity with limited approximation ability, but the combination of several such building blocks leads to complex motions that emerge from the execution of a sequence of these primitives. The contribution of this chapter is threefold: First, a monolithic neural network approach is described that stores efficiently movement primitives. This network can produce smooth complex movements by sequencing the learned movement primitives. Second, the memory capacity of this monolithic approach is evaluated extensively. And the third contribution is that the global representation in this approach is exploited to blend two known movement primitives to generate new intermediate movements without additional control overhead. The algorithms of this section elicit further work to address the problem of autonomous sequencing of complex motions.

CHAPTER 6

CREATION AND REFINEMENT OF MOVEMENT PRIMITIVES BY CO-ARTICULATION

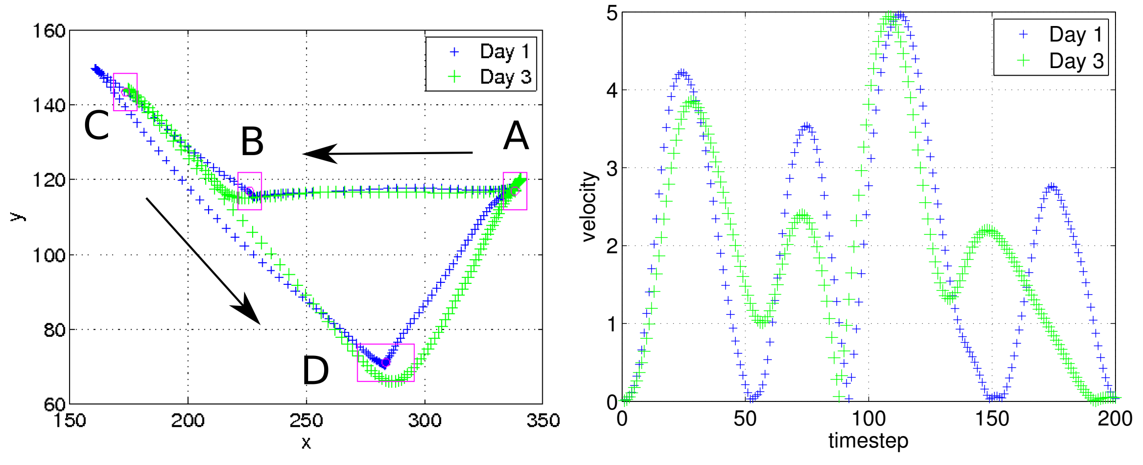
In the previous chapter, the ELM approach is emphasized to be beneficial in the DMP motion generation paradigm, if either multiple motion patterns need to be stored in one representation or superposition of known primitives is necessary. The means to store and execute movement primitives are established, but the learning is organized in a supervised fashion following strictly the imitation learning paradigm. That means each movement primitive is selected directly from the demonstration and implemented in the motion generation method.

In this chapter, a concept to create new movement primitives and to refine a movement primitive is introduced. This is achieved by a semi-supervised learning method which uses self-generated training data for further learning steps. The learning approach utilizes a global encoding together with signal depending output noise to refine the movement primitive over multiple learning steps. The learning process is shown on via-point tasks and the results are compared to the minimum jerk model. It turns out that human-like motion features are incorporated into the learned movement primitives. Early ideas of this work were presented in [Lemme et al., 2012].

6.1 Notion of co-articulation

In our daily lives, people perform tasks that require complex motions such as drawing or handwriting. These abilities are acquired over the years and get more and more refined, if these skills are used on a regular basis (i.e. training and practice). In [Breteler et al., 2003; Sosnik et al., 2004, 2007] human drawing motions were analyzed and the notion of co-articulation was described, where a new movement primitive can emerge with extensive training. This process may start with simple straight motions, which are eventually substituted with more complex, curved primitives. Co-articulatory effects are not only known in drawing movements but also in speech production [Fowler and Saltzman, 1993], where overlapping neighboring phonemes are observed. The question is if such co-articulation effects can be modeled and be used in a movement learning process with the emphasized motion learning methods.

A typical approach to optimize movement primitives with respect to certain criteria are often discussed in the field of reinforcement learning [Schaal et al., 2003b; Peters and Schaal, 2008; Stulp and Schaal, 2011]. Multiple roll-outs are used to explore the state space and minimize specific cost functions modeling the desired behavior. A roll-out defines the process of applying the motion generation model with the current parameter configuration to the task. The trajectory data is recorded and is used for further optimization.



(a) (b)

Fig. 6.1: Conducted via-point task with human participants. The motion starts at point *A* and connects *B C D* (left). The via-points are *B* and *D*. The corresponding speed profile is shown to the right. Note that the via-point *B* is passed at timestep $t = 59$ and the via-point *D* at timestep $t = 140$.

From a biological point of view there is evidence that motor control is affected due to predominant noise signals in the nervous system as shown in [Schaal and Schweighofer, 2005]. This noise signal seems to be dependent on the distribution of the neuron activities. In [Harris and Wolpert, 1998; Wolpert and Ghahramani, 2000; Tresch et al., 2006], the task optimization in the presence of a signal-dependent noise model is proposed. This approach predicts an optimal feed-forward trajectory under the condition of motor commands plus noise and minimizes the target error constraint with zero velocity at the start and end point. In [Stulp and Schaal, 2011], a hierarchical reinforcement learning approach is proposed, which uses multiple uncertainty distributions to optimize the shape properties and target point location, accordingly to a given cost function.

The contribution of this chapter is to propose a flexible learning setup using semi-supervised learning techniques to model co-articulation effects. The concept of semi-supervised learning or self-learning is mainly known in the classification and object detection domain [Nigam and Ghani, 2000; Rosenberg et al., 2005], where e.g. a classifier is trained at first with a small training data set. Then this classifier labels new data samples, which are used again to train the same classifier. This learning setup is able to generate biologically plausible motions by incrementally learning from multiple roll-outs, based on co-articulation of basic movement primitives. The influence of simple signal-dependent output noise is investigated and the resulting learning behavior is shown in [Sosnik et al., 2007]. It is shown that the desired behavior can be modeled by using a global encoding represented by an extreme learning machine (ELM).

6.2 Noise-induced learning process for Dynamic Movement Primitives

The learning progress of a human subject that has been training for three days is shown in Fig. 6.1[1]. Fig. 6.1(a) shows the setup of via-points and the produced trajectories from day one and three. In Fig. 6.1(b) the corresponding speed profiles are shown. Note that at the first day the produced motion is stopping at each via-point, but if training proceeds the via-points *B* and *D* are passed with non zero velocity.

[1]Many thanks to Ronen Sosnik, who provided the data from [Sosnik et al., 2007], which is used to generate this plot.

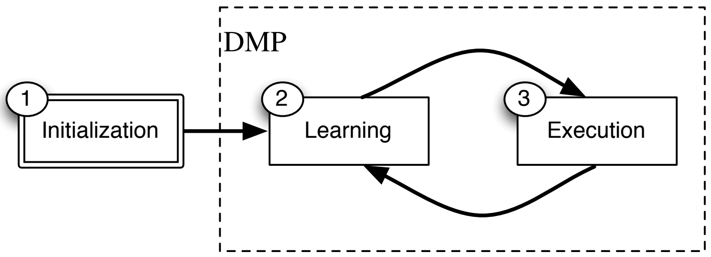


Fig. 6.2: The semi-supervised learning process starts with the initial data set of two articulated trajectories (1). This initial training data is used to create a DMP representation (2). In (3) the learned DMP is used to generate further training data which is again used in (2) to refine the same DMP.

6.2.1 Dynamical movement primitives with signal depending noise

The representation of a movement primitive is modeled with two different variations of the DMP approach. The difference of the two variations is how to approximate the perturbation f, which is added to the spring damper system according to Eq. (5.1). Two different encoding methods are considered for this learning problem: First a local encoding is used represented by a set of Gaussian basis functions (GBFs). Second is the global encoding represented by ELM as used in Chapter 5. For both learning approaches the regression problem can be solved with the OS-ELM learning technique using a initial learning phase and a sequential learning phase as explained in Eq. (5.10) - Eq. (5.12) to determine $\mathbf{W}^{out}$ after each new batch k of training data. The training data-set $\mathbf{D}_k$ is given by $\mathbf{D}_k = (\mathbf{S}_k, \mathbf{Y}_k) = (s(n), \mathbf{y}(n)) : n = 1 \ldots N_{\text{tr}}$ for inputs $\mathbf{S}$ and the corresponding output targets $\mathbf{Y}$. The matrix $\mathbf{H}_k(\mathbf{S}_k) = (\mathbf{h}(s(1)), \ldots, \mathbf{h}(s(N_{\text{tr}})))$ collects the hidden layer states obtained from $\mathbf{D}_k$.

The state representation $\mathbf{h}$ for the GBFs and the output $\hat{\mathbf{y}}$ is given by:

$$f_i(s) = \hat{y}_i = \frac{\sum_{j=1}^{R} \Phi_j(s) \mathbf{W}_{ij}^{out}}{\sum_{j=1}^{R} \Phi_j(s)} = (\mathbf{h}(s)\mathbf{W}_i^{out}), \tag{6.1}$$

with $\Phi_i(x) = exp(-l_i(x-c_i)^2)$ representing the original used non linearity in the DMP approach. c_i and l_i are usually fixed by the designer for all $i = 1, \ldots, d$, where d is the output dimension. A linear distribution of c between 1 and 0 in the experiments is chosen. Due to linear dependence of f on $\mathbf{W}^{out} \in \mathbb{R}^{d \times R}$ it allows to apply any linear regression technique. The ELM network is again given by:

$$\mathbf{h}(s) = \varphi(\mathbf{w}^{inp} s + \mathbf{b}), \tag{6.2}$$

$$f(s) = \hat{\mathbf{y}} = \mathbf{h}(s)\mathbf{W}^{out}, \tag{6.3}$$

where $\mathbf{b}$ are the activation of the bias neurons and $\varphi(x) = 1/(1+\exp(-x))$ denotes the activation function applied to each neuron in the hidden layer with size R. The input is connected to the hidden layer through the input matrix $\mathbf{w}^{inp} \in \mathbb{R}^R$ which remains fixed after random initialization.

The standard regression model $y = g(x) + \varepsilon$ does assume that the random noise parameter ε is chosen from an independent distribution. In the proposed movement learning paradigm, a signal depending noise model is suggested. The noise distribution is linked directly to the input distribution by measuring the neuron's activity. To incorporate signal depending noise the state harvest is organized as follows:

$$\hat{\mathbf{h}}(s) = \mathbf{h}(s) + \varepsilon(s) \tag{6.4}$$

where ε is a random vector drawn from a multi-normal distributions with zero mean and standard deviation $\theta \in \mathbb{R}^R$, with $\theta = \sigma \mathbf{std}(\mathbf{H}_k(\mathbf{S}_k))$. The function $\mathbf{std}()$ gives the standard deviation per neuron and σ scales the amount of signal depending noise.

6.2.2 Semi-supervised learning process

The artificial semi-supervised learning process is separated into three different phases as illustrated in Fig. 6.2. In Fig. 6.2(1), the initialization uses a movement primitive that produces straight lines to connect the three points in the order $\overline{AB}$ and $\overline{BC}$ for M roll-outs. The used set of straight motions have a

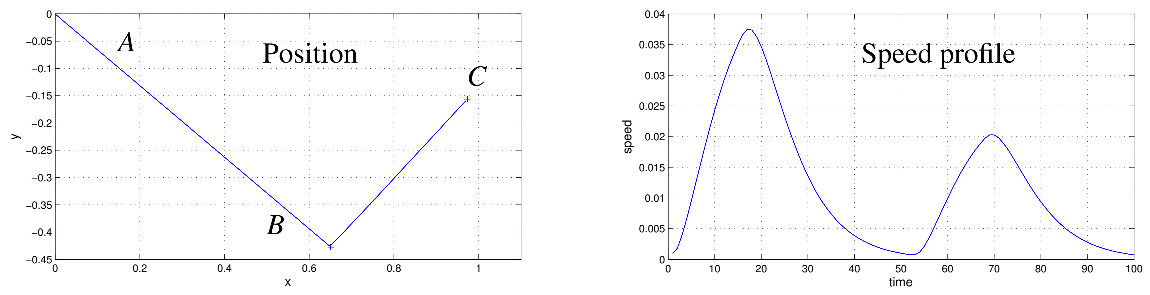


Fig. 6.3: Initial execution phase of two straight lines connecting $\overline{AB}$ and $\overline{BC}$ in this via-point task. The reproduced trajectories (blue) are shown together with their respective speed profile.

bell shaped velocity profile which can be seen in Fig. 6.3. There are different possible motion generation models capable of producing such straight motions with the described features, however, because of consistency the DMP is chosen to represent the motion as a dynamical system. The DMP transformation system is given in Eq. (5.1) and is coupled with a canonical system Eq. (5.2) where initially no perturbations are applied to generate a straight line. In a next step a new movement primitive is formed with initial training data Fig. 6.2(2). The training data is represented by a concatenated trajectory of two straight lines $\overline{AB}$ and $\overline{BC}$. Each trajectory consists of $t = 100$ time steps.

After the learning phase an exploration phase (Fig. 6.2(3)) is applied, where the trained DMP is used to generate M roll-outs. Each roll-out connect A and C, because it is only one DMP representing the motion the via-point is encoded intrinsically. These trajectories are recorded and used in the next learning phase (Fig. 6.2(2)). One epoch comprises one learning phase and one exploration phase (Fig. 6.2(2,3)). Each roll-out of the dynamical system is constrained to have a constant motion duration.

6.3 Co-articulation in a via-point task

The first setup is inspired by the setup shown in Fig. 6.1(a), but simplified. It is assumed that only one primitive is emerging from this task. This setup allows to conduct an indirect qualitatively comparison of the learned new primitives to velocity profiles from human data by using the minimum jerk model as baseline. In the generated data-sets the via-point is located at the extrema of each shape, which is reached approximately at $\tau = 0.5$ of the motion duration. The simplified task setup is shown in Fig. 6.4.

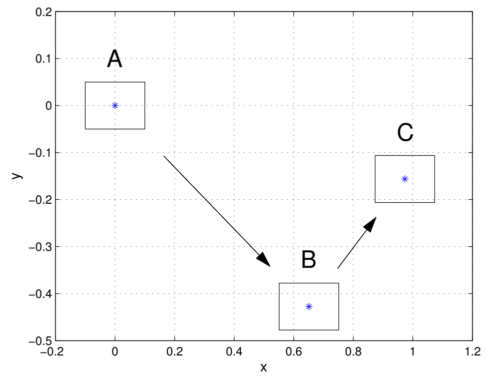


Fig. 6.4: Task with single via-point B. The points need to be connected in the order $\overline{ABC}$.

6.3.1 Baseline comparison for evaluation

The minimum jerk model (MJM) [Flash et al., 1992] plans a trajectory starting from a given start point A to a given end point C through a via-point B with specific timing constraints. The constraint for planning the trajectory is to be as smooth as possible (minimum jerk), start and end with zero speed and a specific time when to pass through the via-point. To plan a $2D$ minimum jerk motions this cost function is minimized representing the rate of change of acceleration integrated over the full motion:

$$\text{Cost} = 1/2\int_{t_1}^{t_2}\left[\left(\frac{d^3x}{dt^3}\right)^2 + \left(\frac{d^3y}{dt^3}\right)^2\right]\mathrm{d}t, \tag{6.5}$$

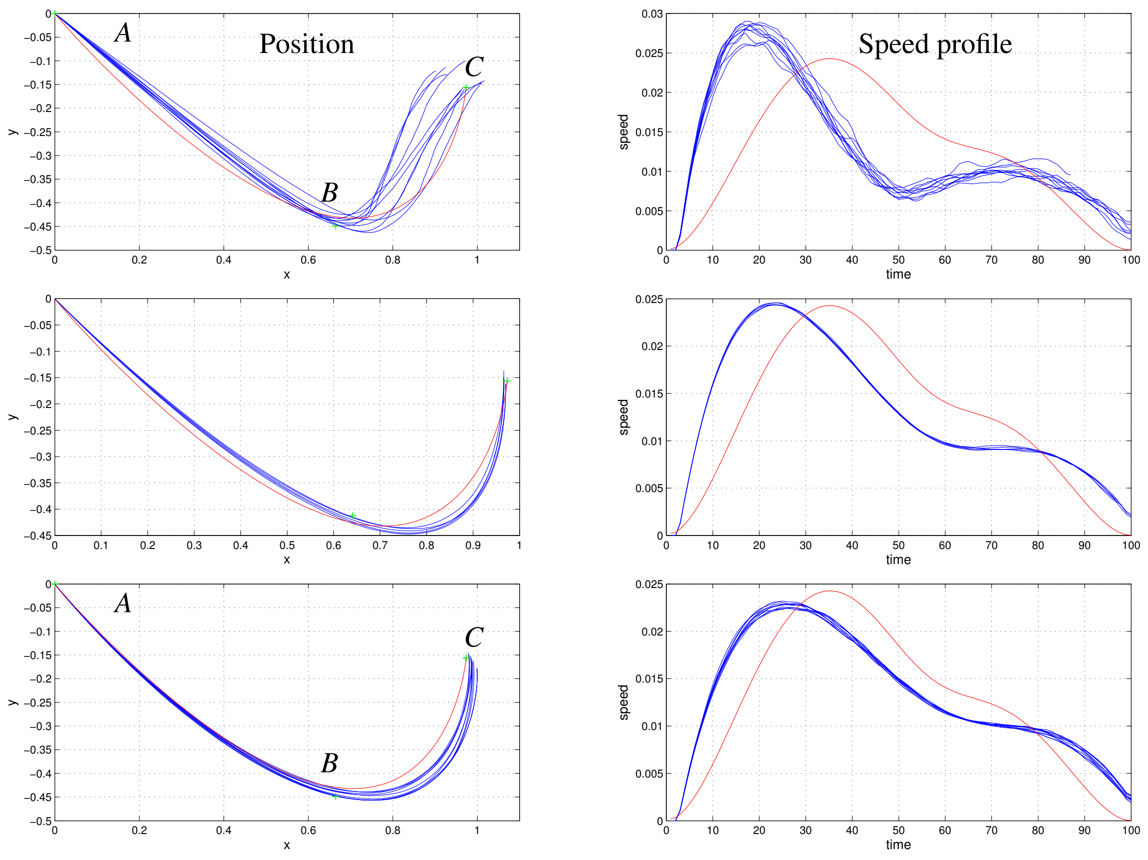


Fig. 6.5: Via-point task conducted with feed-forward neural network ELM by using signal depending noise. The reproduced trajectories (blue) are shown together with their respective speed profile. For comparisons reason the minimum jerk trajectory is given (red) with motion duration $t = 100$ and the constraint that the trajectory needs to pass through the via-point at $t = 50$. The fist row shows the result after 10 iterations. The last two rows show the results after 500 and 10000 iterations. Note that the variation of the trajectories per iteration is reduced over the learning process.

where x and y are the Cartesian coordinates and $t_2 - t_1$ is the motion duration. It is shown that the trajectories modeled by the minimum jerk match the empirical observed features of the human movements, which result from human training process [Sosnik et al., 2007]. However, the MJM does not model the learning process itself.

6.3.2 Properties of newly learned primitives in a single via-point task

In the following experiments constant values for the DMP transformation system (see Eq. (5.1)) are used with the stiffness parameter $K = 7$ and the damping parameter $D = 2.6458$. The GBFs are initialized with a linear distribution of $c_i \in [1,0]$ where $i = 1,\ldots,R$ with $R = 15$. The Gaussian variance is 0.1 for each kernel. During the iterative learning process the trajectories are monitored while the learning proceeds and the trajectory evolution is compared to the planned MJM. The ELM is initialized with $R = 15$ neurons in the hidden layer. The biases b_i and components of $\mathbf{w}^{inp}$ are initialized randomly drawn from the uniform distribution on $[-1,1]$. The first experiment is evaluating the long term properties of the learning process. A three point task with start-point $A = (-1,0)$, end-point $C = (0,0)$ and via-point $B = (0.6510, -0.4276)$ is applied. Fig. 6.5 shows the iterative learning dynamics of the movement primitive over 10000 epochs, where each epoch used $M = 10$ roll-outs. As can be seen in Fig. 6.3 the single trajectories (blue) connecting A, B and C in straight lines, become one single smooth motion shown in Fig. 6.5, which indicates the typical co-articulation effect. The same behavior can be observed in the speed profiles in Fig. 6.5(right column). The result of this experiment is not intuitive because, first,

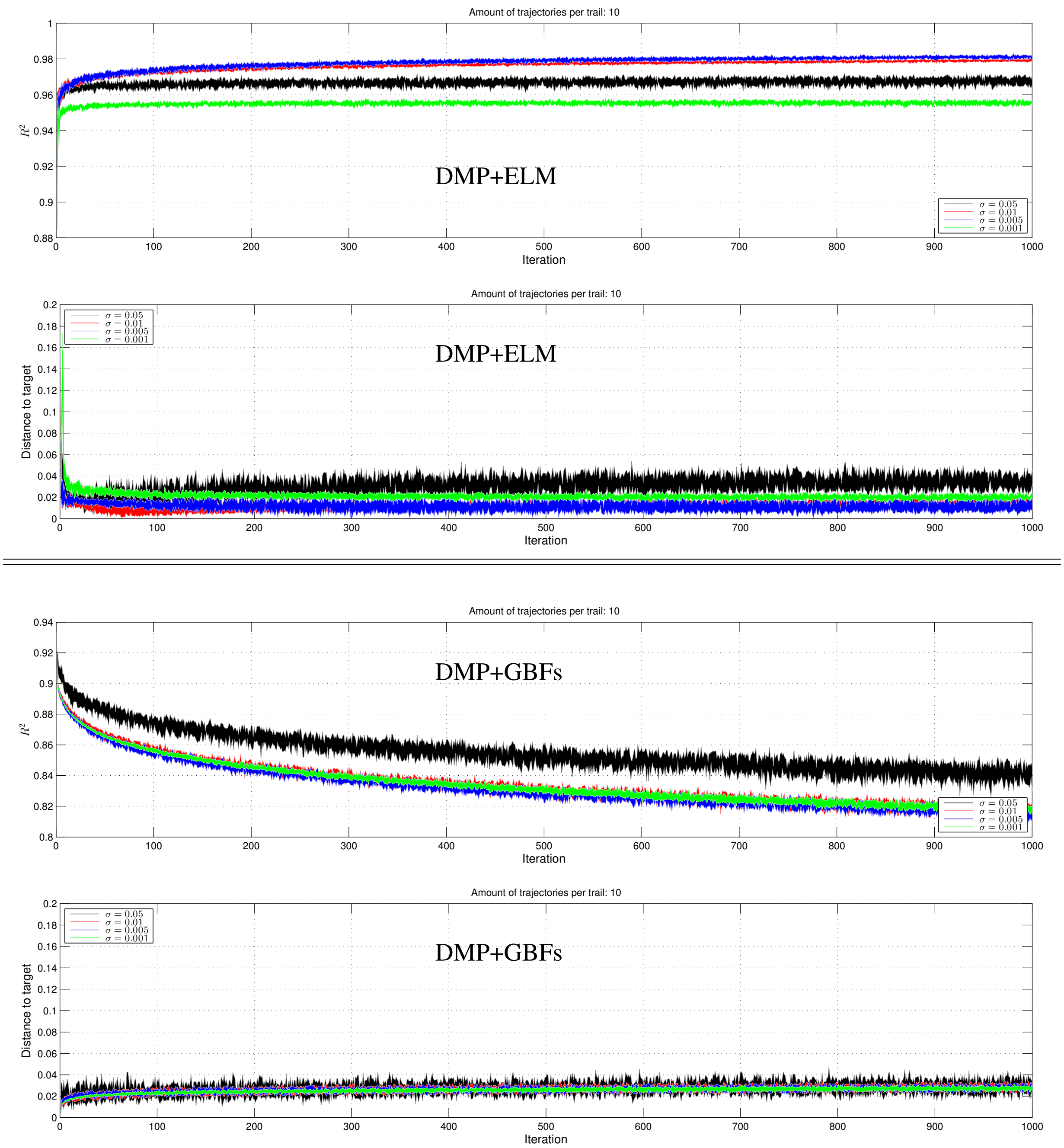


Fig. 6.6: Learning behavior over 1000 epochs and using $M = 10$ roll-outs per epoch. The reproduced trajectories are compared to the minimum jerk model trajectory by the R^2 measure (Top). Note that $R^2 = 1$ would be a perfect match. The second plot shows the distant to the target point C. Each plot shows the learning result for four different output noise levels σ. The last two plots show the same learning process only learned with GBFs.

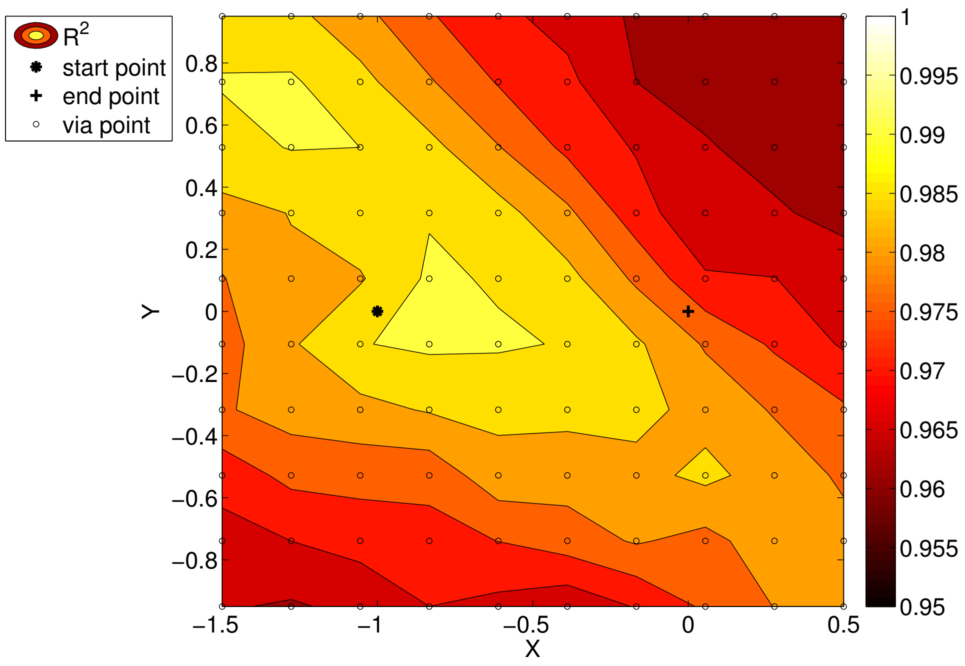


Fig. 6.7: Result of the evaluation processes after 100 iterations for different via-points (circles). The used output noise level $\sigma = 0.005$ hidden layer size 15 regression parameter 10^{-10}.

the resulting shape is converged to one single solution and second the via-point position is preserved and the timing of passing through this point varies only marginal.

Fig. 6.6 shows a quantitative presentation of the learning process for DMP with ELM and GBFs, evaluating the shape compared with the MJM and the target distance at the end of the motion. The iterative learning process using the ELM converges close to the minimum jerk motion with up to $R^2 = 98\%$, whereas the learning process using GMMs seems to diverge from this solution. In the beginning of each learning process the target accuracy is low, however, as learning proceeds the accuracy increases, but it reaches the target point C not perfectly due to the noise induced into the system. Note, that the convergence of the DMP with ELM approach converges close to a minimum jerk motion, which indicates the co-articulation effect of the learning process in this movement primitive representation.

In Fig. 6.7, the learning process is systematically tested for a grid of the via-point locations. A grid of 100 via-points located in $\Omega = [0.5, -1.5] \times [0.95, -0.95]$ is used and measured the similarity of the evolved motion to the minimum jerk model with the R^2 criteria. The ELM is initialized with 15 neurons in the hidden layer and a regression parameter of 10^{-10}. The learning result illustrated in Fig. 6.7 is after 100 iterations and the mean of 5 trails. In the chosen area all of the learned motion explain over 96% of the minimum jerk model. It shows that via-point setup has only little effect on the learning process that leads to the co-articulation effect.

The difference in the learning process between GBF and ELM is interesting and is worth to look at it closer to identify the reason behind the difference. In this experiment 1000 epochs are executed using $M = 10$ roll-outs in each epoch. First the focus is at the timing information of the two approaches. The closest point of the trajectory to the via-point B is measured and is defined as “estimated” via-point. Tab. 6.1 shows the timing information when the via-point is passed in relation of the whole motion duration for each tested noise level $\sigma \in 0.05, 0.01, 0.005, 0.001$. Overall tested noise levels the DMP with ELM can approximate the intended timing of when to pass the via-point, whereas the DMP with GBF is faster at the via-point with as intended.

6.3.3 Scaling to multi-via-points

In the literature, standard via-point tasks are available, which are used to test the movement behavior of humans under various conditions. Two settings are used to test if the proposed approach can scale to more than one via-point.

The initial data-set consists of $M = 10$ roll-outs, where each roll-out is a sequence of straight lines connecting the start point A and end point A through the three via-points B, C and D (i.e. $\overline{AB}$,$\overline{BC}$,$\overline{CD}$ and $\overline{DA}$). The stiffness parameter is set to $K = 7$ and the damping parameter to $D = 2\sqrt{K} = 5.2915$. The ELM is initialized with $R = 15$ neurons in the hidden layer. Fig. 6.8 shows the resulting trajectories and speed profiles depending on the noise level. Again 1000 epochs are used where ten roll-outs per epoch are executed. Three noise levels are tested: $\sigma = [0.01, 0.001, 0.0001]$. From this experiment it becomes clear that the noise level is a significant factor controlling the trade-off between the smoothness of the trajectories and the accuracy.

6.4 Discussion

Learning motions is always difficult, if the task is unknown. This problem is known as the “what to imitate” problem [Nehaniv and Dautenhahn, 2000]. A typical way of how to handle these issues is solved by incorporating prior knowledge. This can be achieved either by human robot interaction as it is done in many imitation learning scenarios or by incorporating constraints like accuracy to the target and shape or even more complex task encoding cost functions as done in reinforcement learning for various tasks.

In this work a semi-supervised learning paradigm is used, which uses self-generated trajectories to optimize the movement primitive representation. It is known that this kind of learning can have major downsides, if e.g. ambiguities are in the solution space. An example for this behavior is learning the inverse kinematics of robots [Sanger, 2004]. One way to solve this problem, at least for inverse kinematics, is goal babbling [Rolf et al., 2011], which uses specific assumptions for the exploration to prevent the learning to suffer from non-convex solution sets and drifting redundancy resolutions. The interesting fact is that in this proposed bootstrapping setup, this form of drifting is not observed if signal depending noise is used. This is surprising, because the learning does not have any additional information about the via-point or the desired shape, however, it still converges to a good solution.

The co-articulation effect observed in Sosnik et al. [Sosnik et al., 2007] is bound to certain conditions which need to be fulfilled. For example, it is crucial that the accuracy to pass through the via-points needs to be relaxed, otherwise the human subjects fall back to straight lines to connect the points. Therefore, it can not be claimed that this learning method represents the only mechanism which does optimization in the motion learning apparatus. Especially, if visuo-motor coordination is taken into account, a combined control strategy seems to be plausible and furthermore interesting to look at in future work.

Model / Noise	$\sigma = 0.05$	$\sigma = 0.01$	$\sigma = 0.005$	$\sigma = 0.001$
DMP + ELM	0.45 ± 0.004	0.45 ± 0.003	0.46 ± 0.005	0.46 ± 0.005
DMP + GBF	0.43 ± 0.053	0.35 ± 0.006	0.35 ± 0.006	0.36 ± 0.006

Tab. 6.1: Timing information specifying when the trajectory passes through the via-point B, for both approaches DMP with ELM and GBF. Note that the timing is not significantly different over the different noise levels, however, the DMP approach using GBF reaches the via-point earlier. The initial timing is set to 0.5.

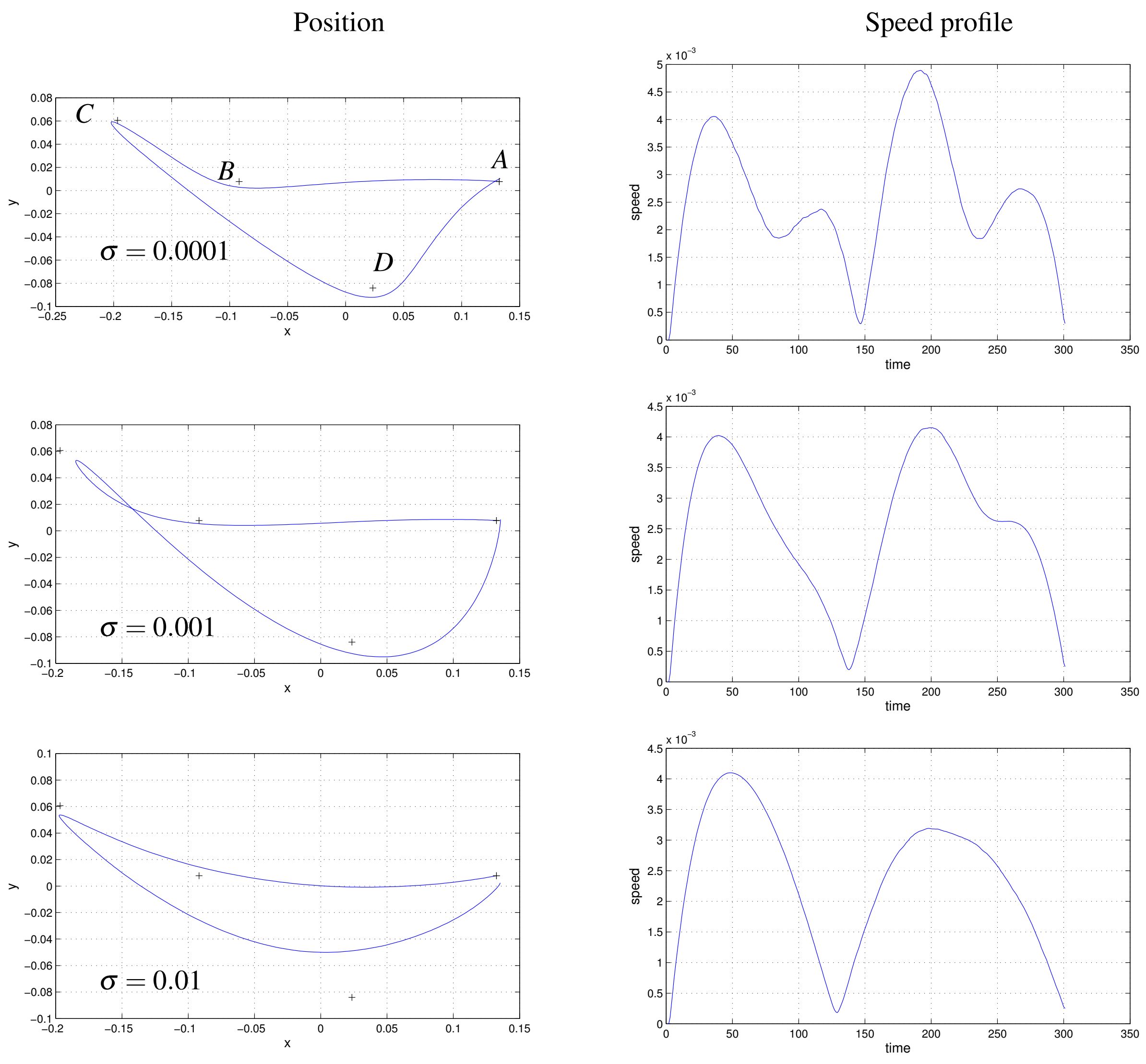


Fig. 6.8: Learning behavior over 1000 epochs and using $M = 10$ roll-outs per epoch. Results after learning with three different noise parameters. The spatial trajectory is illustrated in the left column and the speed profile in the right column. (The noise level is in the bottom row $\sigma = 0.01$, in the second row $\sigma = 0.001$ and in the top row $\sigma = 0.0001$. Note that by increasing the noise level in the learning paradigm the resulting trajectory becomes more smooth but less accurate according to the via-points.

6.5 Conclusion

In this chapter a novel semi-supervised learning process is presented to model a similar learning process of humans specified by the notion of co-articulation. The learning behavior is demonstrated in different via-point tasks. Given an initial set of straight motions, the learning proceeds creating a new movement primitive, which is used for movement exploration to further evolve the new primitive.

A variation of the DMP representation is used together with an OS-ELM network (described in Chapter 5) to learn the movement primitive. The learning method is extended to use biological plausible signal-dependent noise. Using signal-dependent noise is beneficial for iterative learning. It converges to a minimum jerk trajectory. In a systematic comparison of the DMP using the ELM encoding with the DMP approach using GBFs, it is shown that the global encoding of the ELM is necessary for convergence of the iterative learning process. Additionally, the learning process is evaluated for different locations of the via-point to demonstrate the robustness of the learning paradigm. Also the learning scenario is extended from a one via-point task to a multi via-point task.

The results presented in this chapter establish the necessary basis for further investigations in larger skill architectures where motion primitives need to be refined.

CHAPTER 7

PERCEIVING KNOWN MOVEMENT PRIMITIVES IN COMPLEX MOVEMENTS

In Chapter 3 and Chapter 4, the differences of state-of-the-art motion generation algorithms suitable for movement primitive representations are discussed. The descriptions were focused on plausibility to human generation performance and on safety issues regarding real world robot applications in form of stability features.

Articulated movements are fundamental in many human and robotic tasks. While humans can learn and generalize arbitrarily long sequences of movements, and particularly can optimize them to fit the constraints and features of their body, robots are often programmed to execute precise but fixed point-to-point patterns. In this chapter, a novel approach is proposed to interpret and reproduce articulated and complex trajectories as a set of known movement primitives. Instead of achieving accurate reproductions, the proposed approach aims at interpreting data in an agent-centered fashion, according to an agent's movement primitives. In particular, because trajectories are understood and abstracted by means of agent-optimized primitives, the method has two main features: (i) reproduced trajectories are general and represent an abstraction of the data, and (ii) the algorithm is capable of reconstructing highly noisy or corrupted data without pre-processing thanks to an implicit and emergent noise suppression and feature detection. This study suggests a novel bio-inspired approach to interpreting, learning and reproducing articulated movements and trajectories. The conceptional ideas and results of this chapter are also published in [Soltoggio et al., 2012; Soltoggio and Lemme, 2013].

7.1 Using the agent's own movement capabilities for perception

Humans and animals are capable of learning, perfecting and reproducing complex trajectories that allow them to perform a variety of tasks. Following the concept of motor resonance it motivates that using the own motion representation to perceive and represent observed motion behavior becomes plausible, which is shown even in human infant learning behavior (see [Berthier, 1996] and [Paulus, 2014] for a review).

As shown in the previous chapters a good representation for motions in robotics are movement primitives. One intrinsic feature of motion primitives is that they generate basic and general movements that can then be combined to compose arbitrary long and convoluted trajectories. While biological studies continue to reveal the neurological bases of motion primitives [d'Avella et al., 2003b; Hart and Giszter, 2010], computational models provide examples of computer generated primitives as discussed in Chapter 2. Fundamental questions that arise with the use of primitives are: what are the features of a set of primitives? How are primitives composed to perform articulated movements? And what role do they play in interpreting, coding and learning complex movements?

Two categories of decomposition approaches can be identified. First, the data-centered approaches, which analyze the data to find features, e.g. inflection points, points of discontinuous derivative, critical points,etc. And second, the agent-centered approach [Atkeson and McIntyre, 1986] implies that the algorithm does not analyze directly the demonstrated trajectory as opposed to other approaches [Kohlmorgen and Lemm, 2001; Hellbach et al., 2009; Mohan et al., 2011]. Instead, the demonstrated trajectory is approximated by means of a process of learning-by-doing in which the performance, or the accuracy of a reproduction, is improved over time with increasingly refined decompositions. Avoiding the analysis of demonstrated data results in two main features of the algorithm. The first is that a reproduction of a demonstration is biased by the agent's set of primitives. In this respect, the reproduction represents an *interpretation* of a demonstration. In other words, any demonstration, which was generated by an unknown process, is being fitted with the agent's fixed primitives. While this may appear as a limitation, it also means that no assumptions on the demonstrated trajectory are required. The agent attempts to achieve a best approximation with its current primitives used as a tool to interpret input data. A second feature is that the algorithm may take as input highly noisy and corrupted data and displays implicit noise suppression and feature detection. In Sec. 2.3.2, related approaches are summarized that try to identify motion primitives in trajectories.

The aim of this work is to achieve an agent-centered interpretation and progressive adapting that fits in the first place the robots' capability, as opposed to a data-centered decomposition analysis. One interesting aspect of the proposed method is that, similarly to [Wada and Kawato, 1995, 2004; Rohrer and Hogan, 2003], the decomposition of complex trajectories initiates as a rough approximation based on one single movement primitive. Interestingly, a complex trajectory encompassing many convoluted parts, e.g. a handwritten long word, is unlikely to be adequately represented by one single stroke. Yet, by adopting this counter-intuitive approach, a fundamental step in an iterative process can be achieved towards further and more precise decompositions. Points of decomposition are progressively discovered during the iterative process. At each iteration, the part of the reproduction with the maximum discrepancy with the demonstrated trajectory is considered for improvement. Thus, segmentation points are introduced with the simple but effective heuristic of observing the point of maximum error. Decomposition points can also be later suppressed if more general primitives are discovered to fit a part of the demonstration. The deletion of segmentation points is a bio-inspired search that, once some main features of a demonstration are captured, it relaxes constraints to find better solutions and overcome local optima.

It is important to note that the present algorithm only focuses on geometrical properties of the trajectories, while is agnostic to the velocity profiles. This apparent limitation in reality allows for a more flexible interpretation of trajectories, which may not be necessarily determined by the velocity profile used during generation. Once more, no assumption on the demonstration implies that any demonstration can be observed, decomposed and reproduced with the proposed algorithm. Tests on both geometric and human generated trajectories reveal that the use of own primitives results in remarkable robustness and generalization properties of the method.

7.2 Methodology for decomposing and interpreting trajectories

This section describes the proposed decomposition algorithm in all its parts, motivates the bio-inspired approach and illustrates all the steps to reproduce the method.

7.2.1 Movement primitive libraries as decomposition tools

The decomposition algorithm requires a set of pre-learned primitives that can be freely chosen and generated by means of a variety of methods. This feature is particularly important to integrate the proposed algorithm with the well established methods for primitive generations cited above. The performance of the algorithm in the decomposition varies in accuracy and approximation according to the movement primitive library, as later tests show. Nevertheless, the method can decompose complex trajectories even with very poor movement primitive libraries.

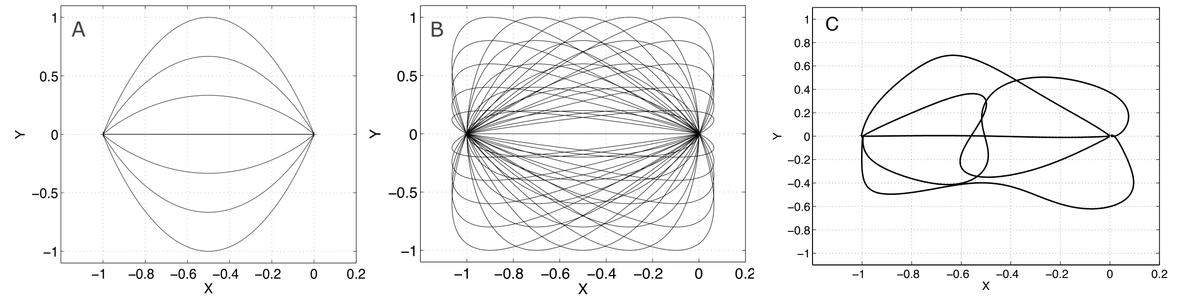


Fig. 7.1: Movement primitive libraries. (A) This small set of primitives was generated with the Minimum-Jerk Model (MJM) [Flash and Hogan, 1985] and contains one straight line and six curved lines. It is referred to as the MJM 7-set. (B) Shows also MJM trajectories. This set, contains 51 primitives both with symmetric and asymmetric geometry (MJM 51-set). (C) This set, composed of only five primitives, was learned from human demonstrations using an Extreme Learning Machine (ELM) [Huang et al., 2004]. Primitives are rotated and scaled to be fitted to the starting and ending point of a demonstration.

To show the influence of different movement primitive libraries, the current study considers three sets: two sets generated with the minimum-jerk model (MJM) [Flash and Hogan, 1985] and one is generated by using a feed-forward network (Extreme Learning Machine) [Huang et al., 2004], which is the NiVF (see Chapter 3) without the additional stability constraints. Stability constraints are not necessary here because no perturbations are applied. The networks were trained to reproduce a set of human drawn trajectories. One minimum-jerk model (MJM) set is composed of seven symmetric primitives (Fig. 7.1A), and a more complex set has 51 primitives with symmetric and asymmetric shapes (Fig. 7.1B). The ELM-set has 6 primitives (Fig. 7.1C). Experiments can be extended to include further sets.

7.2.2 Trajectory matching and iterative decomposition

Regardless of the length and complexity of a given trajectory (demonstration), the counter-intuitive position in this study is that only the start and end points are initially considered as extremities of one single primitive. The agent searches among its own primitives a best match. It is assumed that a primitive can be rotated and scaled to connect the initial (I) and final (F) points of the demonstration. If the demonstration is long and articulated, the first match is inevitably a gross approximation. Four numerical criteria to compare trajectories were considered in the current study: (1) a maximum point-wise error (MPE), (2) a point-wise mean square error (PMSE), (3) the area A between the demonstration and the reproduction, and (4) a measure of parallelism Θ. The four criteria are computed by the following equations:

$$\mathrm{MPE} = max(||x(i)-\hat{x}(i)||) \tag{7.1}$$

$$\mathrm{PMSE} = \frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}(x_j^i-\hat{x}_j^i)^2 \tag{7.2}$$

$$A = \sum_{i=1}^{N-1} F(x(i),x(i+1),\hat{x}(i),\hat{x}(i+1)) \tag{7.3}$$

$$\Theta = \frac{1}{N}\sum_{i=1}^{N-1}\frac{x(i+1)-x(i)}{||x(i+1)-x(i)||}\cdot\frac{\hat{x}(i+1)-\hat{x}(i)}{||\hat{x}(i+1)-\hat{x}(i)||} \tag{7.4}$$

where $x(i)$ is a two dimensional coordinate point of the demonstration, $\hat{x}(i)$ a coordinate point of the reproduced trajectory, M is the dimensionality of the data ($M = 2$ in the current study), N the number of samples, and F gives the area of the tetragon specified by the input arguments. The data points **x** are obtained by cubic splines interpolation of the original sampling to ensure that their normalized distance and N are equal.

Eq. (7.1)-Eq. (7.4) can be used independently or linearly combined to assess how similar two trajectories are. Eq. (7.1)-Eq. (7.3) reach zero for perfectly matching trajectories, while Eq. (7.4) is equal to

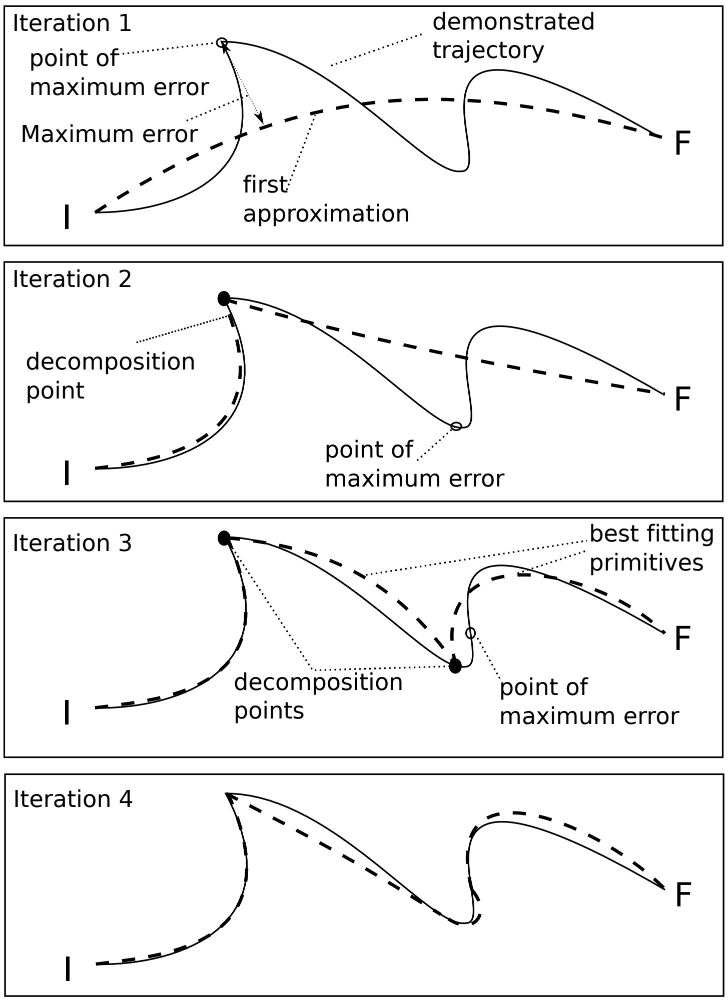


Fig. 7.2: Graphical illustration of the iterative decomposition process. Iteration 1 begins selecting one single primitive as approximation of the global trajectory between the initial (I) and end (E) points. The primitive is chosen applying one of the criteria expressed by Eq. (7.1)-Eq. (7.4). On the demonstration, a candidate segmentation point is chosen according to Eq. (7.1). In Iteration 2, two better fitting primitives are identified, as well as a new point of maximum error. Iteration 3 and 4 show further steps of the iteration. The algorithm may continue to improve locally the approximation until stop criteria are satisfied.

1 for matching trajectories. Visual observation over many examples revealed that deriving a measure of similarities between two different trajectories is not immediate. In effect, evaluating similarities between trajectories may be domain-dependent or even subjective (as also discussed in Chapter 4). The focus of the study is not to compare the effect on the performance of Eq. (7.1)-Eq. (7.4), nor to propose a best criterion. Different matching criteria are proposed here as alternatives which can be chosen to work with the present algorithm. The tests in the current study use by default Eq. (7.2) because it produced predictable segmentations on a large variety of demonstrations. Eq. (7.1) and Eq. (7.3) are also employed in tests to show the robustness of the method. Once a best matching primitive is identified, the point x^* that returns the maximum error in Eq. (7.1) on the demonstration D

$$x^* : MPE = max(||x(i) - \hat{x}(i)||), \forall i \in D \tag{7.5}$$

is chosen as candidate segmentation point. Thus, the first approximation is used to identify a first decomposition point along the demonstration, i.e. a first point to use in an iterative process of further decompositions. Once x^* is identified, each sub-trajectory to the left and to the right is matched with a best primitive. Two cases are now possible: (1) the reproduction with x^* as segmentation point brings an improvement with respect to the matching criteria expressed by Eq. (7.1)-Eq. (7.4); (2) the segmentation does not bring an improvement. In the first case, the *candidate* segmentation point is promoted to *established* segmentation point and the iterative process can continue on each segment. In the second case, the segment is labeled as final and no further segmentation is considered. Fig. 7.2 illustrates the first four steps of the iterative process on an trajectory.

The heuristic that identifies candidate segmentation points is not based on an optimality measure, which is difficult to infer in an iterative process. The attempt is instead that of identifying potentially appropriate points to improve further a reproduction. The underlying idea is that the furthest point on the demonstration from the current reproduction lays potentially in a part of the demonstration that is not correctly represented by the reproduction, and thus requires further segmentation. This simple heuristic proves effective as demonstrated later in Sec. 7.3.

7.2.3 From sequences of points to sequences of primitives

Any trajectory can be represented as a sequence of close points united by straight lines, if decomposed finely. A decomposition that reproduced exactly the demonstration in such a way minimizes the reproduction error. However, such a decomposition merely copies a demonstrated trajectory without generalizing the overall shape of a movement. The problem is effectively twofold: a classification problem (finding the best matching primitive) and an optimization problem (reducing the number of segmentation points). A trade-off between generality, with few decomposition points, and precision, with many segmentation points, is desired and sought [Edelman and Flash, 1987]. As a rule, generality of one solution is accompanied by a residual error with one particular demonstration. For example, a straight line drawn by a human subject is not exactly straight. But even a precise straight line, if observed by means of an analog process like vision, is encoded as a noisy data set which, unless is pre-processed with an arbitrary noise suppression, is not a perfectly aligned sequence of points. Therefore, a residual error between observed and demonstrated trajectory must be accounted in realistic scenarios. The implication is that decomposing a trajectory to minimize the error may lead to a high number of segmentation points. Most algorithms use a error threshold below which the segmentation is considered satisfactory. This problem derives also form the arguable assumption that trajectories have a length but dimensionless thickness. In robotic and real world scenarios, trajectories are both executed and perceived with a certain tolerance. Accounting for such an aspect is a key aspect to avoid over-fitting, unnecessary computation and excessing segmentation.

The method in this study attempts to mimic a trajectory with given primitives that guarantee generality and may be devised to guarantee also efficiency, optimality, or to conform to particular robotic requirements, without necessary minimizing an error measure. For example, the minimum-jerk model used in the current experiments guarantees energy minimization and is biologically plausible [Flash and Hogan, 1985], while the ELM-set uses a neural learning paradigm that reproduces human drawn trajectories.

Precision of primitives and intersections

Instead of considering the error between demonstration and reproduction as stopping criterion, the current algorithm looks at whether the demonstration and the reproduction have *intersections*. If they have at least one intersection, the demonstration is assumed to have further features that need decomposing. If there are no intersections, the current primitive is assumed to be the best approximation: further decompositions may reduce the error but also reduce generality.

Intersections are intended as two trajectories crossing each other: however, two noisy and overlapping trajectories have many local intersections that would not be considered such by a human observer. Thus, to detect significant intersections, the algorithm associates a precision value to the primitives. Such a precision is an index of how thin a trajectory may be with respect to its length. In effect, this parameter may encode the precision of a mechanical arm, or may be adjusted to account for the variance of many samples, if those are executed by imprecise human movements. In short, the precision parameter is a necessary element in the interpretation of an observed trajectory. It answers the questions: what are the agent's perception and execution capabilities? What is a realistic precision to be implemented when reproducing a demonstration?

The case is illustrated in Fig. 7.3 in which the primitive is shown with an associated thickness. If the demonstrated and performed trajectories *do not intersect*, the algorithm infers that there are no further prominent features in the demonstration that need to be reproduced with further segmentations. Non-intersecting trajectories can be somehow distant, but the matching criterion (Eq. (7.1)-Eq. (7.4))

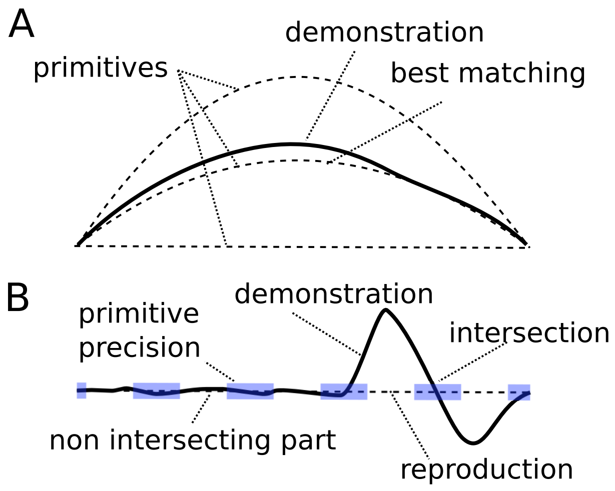


Fig. 7.3: Best fitting and intersections. (A) Three primitives are shown in the attempt to match a demonstrated trajectory. The best matching primitive is not a perfect reproduction, but rather a general abstraction of the demonstration. (B) A demonstration is matched to a straight primitive in which the second dimension, or precision, is shown. The left-most part of the trajectory, although not perfectly straight, does not intersect the two-dimensional primitive, suggesting that no further features are present in the demonstration. The right-most part of the trajectory instead clearly intersects the primitive, indicating the utility of further decompositions.

ensures that this distance is minimized. No intersections mean effectively that the reproduction and the demonstration are as close as possible given the current set of primitives. One exception is when the demonstration exists the reachable area of all primitives. This is for example the case of a circle drawn with a nearly overlapping start and end point. When the demonstration exists the reachable area of the primitives, further segmentations are enforced.

In the experiments of this paper, the smaller sets (MJM and ELM) have a precision value $p = 20 = 1/0.05$, where 0.05 is the thickness of the primitive normalized to the shortest side of the drawing area. The more accurate 51-primitive set has a precision $p = 100 = 1/0.01$, i.e. the thickness of a primitive is 1% of the drawing area. A thickness of 0 corresponds to infinite precision, a concept that does not describe real data from a demonstration and clearly underlines the importance of considering thickness values higher than 0. Higher precision values can be adopted when the demonstration is known to be very accurate. Intersections are detected analytically by computing the cross products between the direction of the primitive and the error vectors of all points: if the cross products have different signs and the error vectors are greater than the line thickness, then an intersection is detected.

The intersection criterion attempts to capture features of the demonstration that are *observable* with set of primitives used. It is nevertheless possible to use a more traditional stopping criterion, for example requiring that the maximum error is decreased below a certain threshold. Such an approach may be used when more emphasis on minimizing the error is necessary and an approximation that respects an error constraint is desired. This variation was experimented in the current algorithm and is easy implementable by letting the algorithm continue the segmentation until the maximum error falls under a threshold. A similar variation may also include, for example, a measure of how parallel two trajectories are (i.e. Eq. (7.4)). The algorithm may be required to continue segmenting until a certain threshold is reached. These variations of the algorithm require more human supervision in setting such an error threshold and understanding what matching criteria are needed in a particular scenario. In some cases, introducing additional matching measurements and stopping criteria may lead to reproductions that are perceived visually as better approximations.

Deleting segmentation points

The iterative process implies that the interpretation of the demonstration (i.e. the solution) varies and improves at each further decomposition. One question is whether decomposition points that were found initially during the process are still good segmentation points later as the accuracy improves. Inspired by theories of motor learning in humans [Bernstein, 1967], the proposed method introduces a type of search that releases early constraints when a new segment is added. At the insertion of a new decomposition point, primitives are searched to the left and to the right of the candidate point. The search may go beyond the immediate left and right segments. It is possible to search further left and further right,

Seg.	Primit.	Start	Scaling	Angle	Final
1	number	xy coor.	factor	angle	y/n
2	..	..	..	..	..

Tab. 7.1: Representation of a trajectory as a sequence of primitives. Segments (i.e. rows in the table) are added and occasionally removed during the iterative process. For each segment, it is necessary to specify which primitive is used (2nd column), the starting point (3rd column), the scaling and angle (4th and 5th column) and whether the segment can be further decomposed (6th column).

thereby attempting larger generalizations. Neighboring decomposition points are eliminated if more general primitives without intersections are discovered.

This check guides the search to avoid local optima, and at the same time it helps reduce the number of overall segmentation points, thereby achieving more general solutions. Releasing constraints implies more computation while searching larger primitives that may skip segmentation points. This type of search is nevertheless far from exhaustive: the further exploration relies on the current segmentation. It represents an attempt to reorganize parts of the trajectory according to new knowledge that was gathered during the iterative segmentation process.

7.2.4 The iterative algorithmic procedure

It is now possible to introduce a flow chart to explain in detail the overall procedure. Fig. 7.4 helps follow the detailed explanation below. The algorithm is also reproducible with the MATLAB code available for download at `http://www.cor-lab.de/decomp`.

The algorithm starts selecting one primitive that best matches the demonstrated trajectory (Fig. 7.4, blocks 1 and 2). The best match is obtained comparing all primitives with the demonstration and choosing the primitive that minimizes a measure of discrepancy (Eq. (7.1) and Eq. (7.3)) or maximizes a measure of similarity (Eq. (7.4)). In the next step (block 3), the algorithm finds the point of maximum error between the demonstration and the reproduction (Eq. (7.5)). This is a candidate segmentation point and is located in a part of the demonstration that is poorly approximated. Initially there is only one segment. As the iterations proceed, more segments are created. When created, each segment is labeled as *non-finalised*, meaning that further decompositions are possible. The point of maximum error is sought on a non-finalised segment (blocks 3 and 4). The algorithm now checks whether the primitive intersects the demonstration or not (block 5). As illustrated in Fig. 7.3A and B, an intersection suggests the presence of a relevant feature that can be captured with further decompositions. If the best matching primitive does not intersect the demonstration (block 5), the demonstration may be laying outside the reachable area of the primitives (block 6). This case, or the case in which there is an intersection, mean that there are additional features in the demonstration that need to be captured. Therefore, the algorithm proceeds with the segmentation (block 7). Otherwise, the current segment is finalized (block 10). The search in block 7 is carried out by exploring primitives that approximate the left and right parts of the demonstration from the candidate segmentation point. Such a search involves also the elimination of older segmentation points if better approximations are found. The search for better primitives in block 7 may or may not result in an improvement of Eq. (7.1)-Eq. (7.4). If no improvements can be achieved, the segmentation point is rejected and that segment is labeled as finalized (blocks 8 and 10). If an improvement is found, the segmentation point and the left and right primitives are promoted as part of the current segmentation (block 9).

Throughout the process, the representation of a demonstration is updated. Tab. 7.1 shows how the primitive-based symbolic trajectory is described. At the first iteration, only one row is present. Further segmentations add more rows describing primitives, start point, scaling and angle, and whether the segment is finalized.

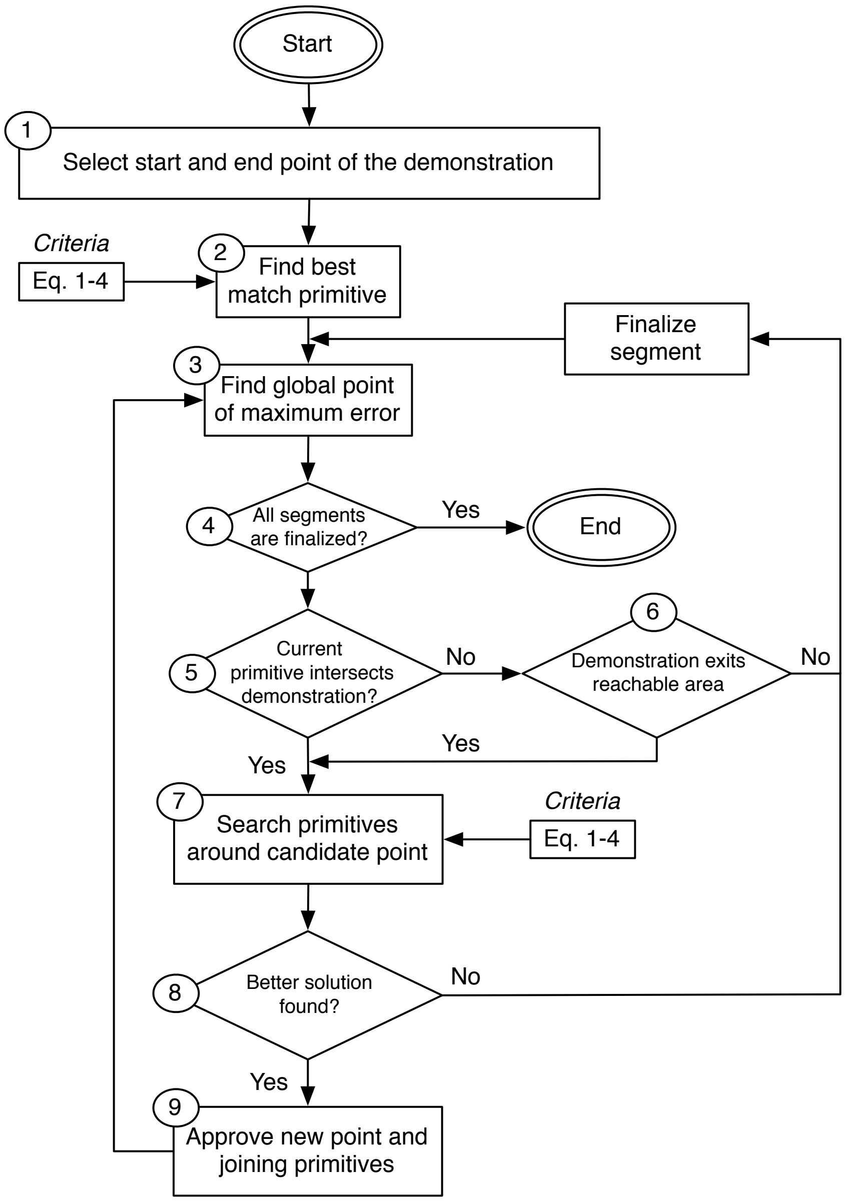

Start
1
Select start and end point of the demonstration
Criteria
Eq. 1-4
2
Find best match primitive
Finalize segment
3
Find global point of maximum error
4
All segments are finalized?
Yes
End
6
5
Current primitive intersects demonstration?
No
Demonstration exits reachable area
No
Yes
Yes
7
Search primitives around candidate point
Criteria
Eq. 1-4
8
Better solution found?
No
Yes
9
Approve new point and joining primitives

Fig. 7.4: Flow charts describing the various phases and iterative nature of the algorithm. The numbers that identify each block are used as references in the text to describe each phase.

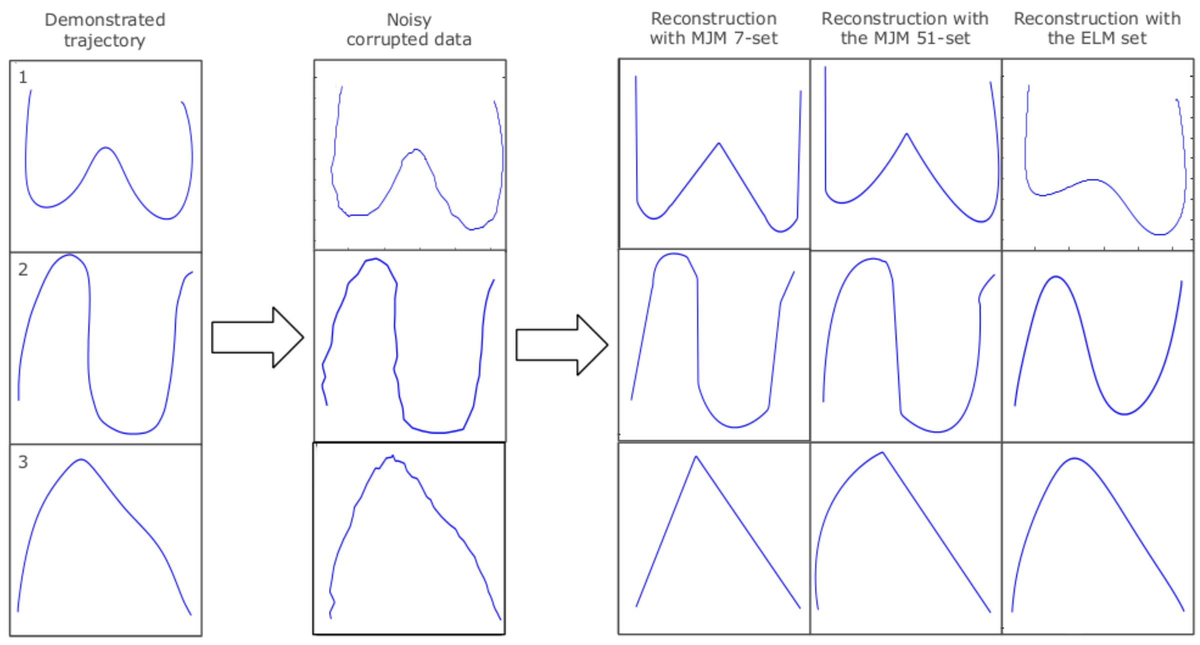


Fig. 7.5: Examples of decompositions of short demonstrations. (First row) A hand-written W (first column) is first corrupted with noise (second column). The noisy data is used to run the algorithm with the 7-set (third row) and with the 51-set (fourth row). Finally, the fifth column shows the interpretation and decomposition with the ELM-set of primitives. The rows below show similar plots for different demonstrations.

7.3 Decomposing complex trajectories with a movement primitive library

This section reports the simulation results of the algorithm applied to a variety of demonstrated trajectories, from simple to complex. A fundamental aspect of the proposed decomposition algorithm is that it utilizes a pre-learned movement primitive library. In the following the impact of different movement primitive libraries for a variation of complex trajectories is demonstrated.

7.3.1 Reconstructing short demonstrations

The decomposition algorithm is applied here on human and machine generated trajectories affected by noise. These basic examples have the purpose of showing how the algorithm interprets and reconstructs short noisy trajectories. Fig. 7.5 shows the application of the algorithm to three different demonstrations after they were corrupted with noise. The method favors elegant decompositions with few primitives, resulting in some cases in a residual error between demonstrated and reproduced trajectories. The discrepancy stems from the more general trajectories chosen by the agent with respect to the irregular human generated data. The decomposition with the MJM set of 51 primitives appears more accurate in comparison with the decomposition from the 7-set. The small set of 7 primitives instead captures the main features of the demonstrated trajectories favoring straight lines, effectively achieving a higher level of abstraction. The implication is that in front of complex demonstrations, agents or robots with few primitives can nevertheless utilize the algorithm to decompose a demonstration according to their basic skills. The ELM-set, despite having only five primitives, performed very well. The reason is because the ELM-primitives were trained on the same trajectory later presented for reconstruction. Therefore, the ELM-set contains primitives that match well the demonstrated trajectory. Nevertheless, it must be noted that the demonstrations are not exactly the same as the primitives and, moreover, the data seen by the algorithm is the corrupted data in the second row in Fig. 7.5.

From this first test it emerges one important and bio-inspired feature of the algorithm. The method

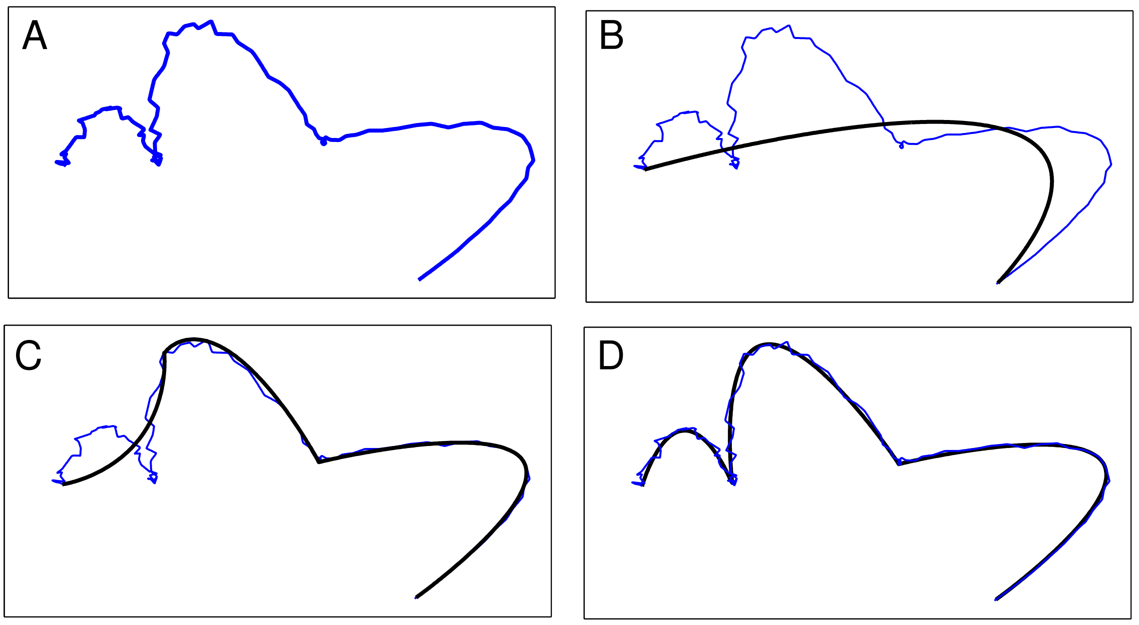


Fig. 7.6: Examples of reconstruction from a noisy machine generate trajectory. (A) The original demonstration affected by noise. (B) The first step of the decomposition. (C) The third step of the decomposition (D) Fourth and final step of the decomposition. The algorithm found the original primitives that were used to draw the demonstration before noise was applied.

appears to reconstruct, in a way to recognize, those trajectories that are similar to the known primitives. The reconstruction by the MJM 51-set in the third row is more abstract and less similar to the original than the reconstruction by the ELM-set, despite the considerably larger library of primitives in the 51-set. However, while the ELM-set performed well in this particular test, the 51-set is more generic and is likely to perform better on other trajectories with arbitrary geometry.

Further tests were performed on automatically generated trajectories with additional high level of noise. Fig. 7.6 illustrates the capability of the algorithm in reconstructing corrupted data. The demonstration was created with the same primitive set used for the reconstruction, which in part explains the correct matching. Nevertheless, this approach appears biologically plausible because humans too tend to recognize in imprecise images objects and shapes that were previously learned [Edelman, 1998]. As hypothesized also in [Wada et al., 1995], the reconstruction and reproduction are closely coupled: the present algorithm shows that noisy data are *recognized* with or *fitted* to the known primitives.

The primitives are executed sequentially without additional procedure to join them. Therefore, points of discontinuous derivative are noticeable where primitives join. Smoothing a trajectory requires the understanding of whether a point is a cuspid, i.e. the trajectory has a discontinuous derivative, or it can be rounded with a co-articulation algorithm [Sosnik et al., 2004; Kulvicious et al., 2012]. As this particular problem was not the focus of the present algorithm, all primitives are joined without blending or co-articulation.

A further test was executed to assess the performance of the algorithm on different trajectories without noise. One hundred trajectories were constructed with sequences of three primitives from the 51-primitive set, applying random primitive lengths and rotations. One example is shown in Fig. 7.7. Similarly to the results in Fig. 7.5, the decomposition with the 7-primitive set produced schematic and abstract interpretations as that in Fig. 7.7B. A straight primitive was frequently used to approximate a demonstration with a low curvature. Alternatively, a combination of a straight and a curved primitive was used to interpret an unknown curvature as in the lower left part of Fig. 7.7B. The analysis revealed

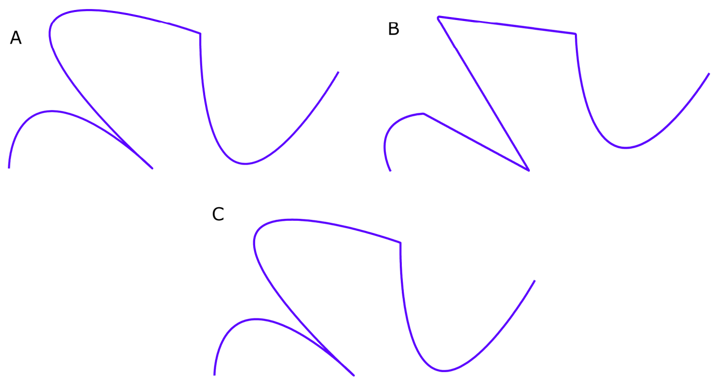


Fig. 7.7: Testing the decomposition on machine generated trajectories. (A) One hundred machine-generated trajectories as this one in the example are used for the extensive performance test. (B) Reproduced trajectory with the small 7-primitive sets. (C) Reproduced trajectory with the large 51-primitive set.

that, although the demonstrations are exact and without noise, the algorithm could not always find the exact primitives that were used to create the demonstration: similar primitives could occasionally be used to create good but not exact reproductions. This is an expected consequence of the fact that the algorithm does not minimize the error between demonstration and reproduction. The test shows that the algorithm expresses its full potential in reconstructing corrupted data (Fig. 7.5) rather than reproducing precise demonstrations.

7.3.2 Decomposition of handwriting

The decomposition of human-generated handwriting trajectories is a task in which the symbolic aspect is more important than the exact geometry. In other words, global features in a trajectory are fundamental in distinguishing different letters more than the precise geometry of the trajectory. The proposed algorithm was shown in the previous section to be suited to extract high level representations from noisy data. It is natural to ask whether this feature may be of use as a step towards abstracting human handwriting. Note that the experiment in this section decomposes and represents handwriting as movement primitive library, but it does not interpret or map trajectories to letters.

Two examples of human handwriting data were analyzed. A first word “Hello” was decomposed as shown in Fig. 7.8. The first row (Fig. 7.8A) is the original trajectory. Fig. 7.8B is the first approximation, i.e. one single primitive. Fig. 7.8C shows the representation after 7 steps. The algorithm has identified some of the main features of the demonstration. Fig. 7.8C illustrates the 9th step, demonstrating how each step is functional in discovering further features. The letter ’h’ and one ’l’ are already readable after 9 steps. Fig. 7.8E shows the final reproduction.

The decomposition proved robust with respect to different trajectory matching criteria (Eq. (7.1)-Eq. (7.3)) and precision parameters. Fig. 7.9 shows the final decomposition of the word “Hello” with various criteria. Fig. 7.9A uses Eq. (7.3) as matching criteria (i.e. the area between trajectories). Fig. 7.9B is a decomposition with trajectory matching criterion Eq. (7.1). And finally, Fig. 7.9C is a decomposition with Eq. (7.2) and a lower precision $p = 20$. The reproductions in Fig. 7.9 are similar but not identical; also compare with Fig. 7.8E in which Eq. (7.2) and precision $p = 40$ were used. It can be inferred that different matching criteria and precision parameters affect the decomposition but do not change significantly the capability of the algorithm to represent a demonstration.

Fig. 7.10 illustrates the decomposition process for the word “Amarsi”, whose original handwriting is plotted in Fig. 7.10A. Fig. 7.10B and C show iterations two and four during decomposition with the 51-primitive set. Also in this case the algorithm begins by reproducing the most relevant features of the demonstrated trajectory. Note that the way the algorithm proceeds is determined by the tentative segmentation points discovered with the criterion of the maximum error between demonstration and reproduction. The hypothesis is that this criterion, although trivial and of simple implementation, is nevertheless effective in finding progressively prominent features of a demonstration. The illustration of the step-wise iteration demonstrates exactly that. Fig. 7.10D shows the final approximation with the large 51-primitive set. Fig. 7.10E shows the reproduced trajectory as it was decomposed by the 7-

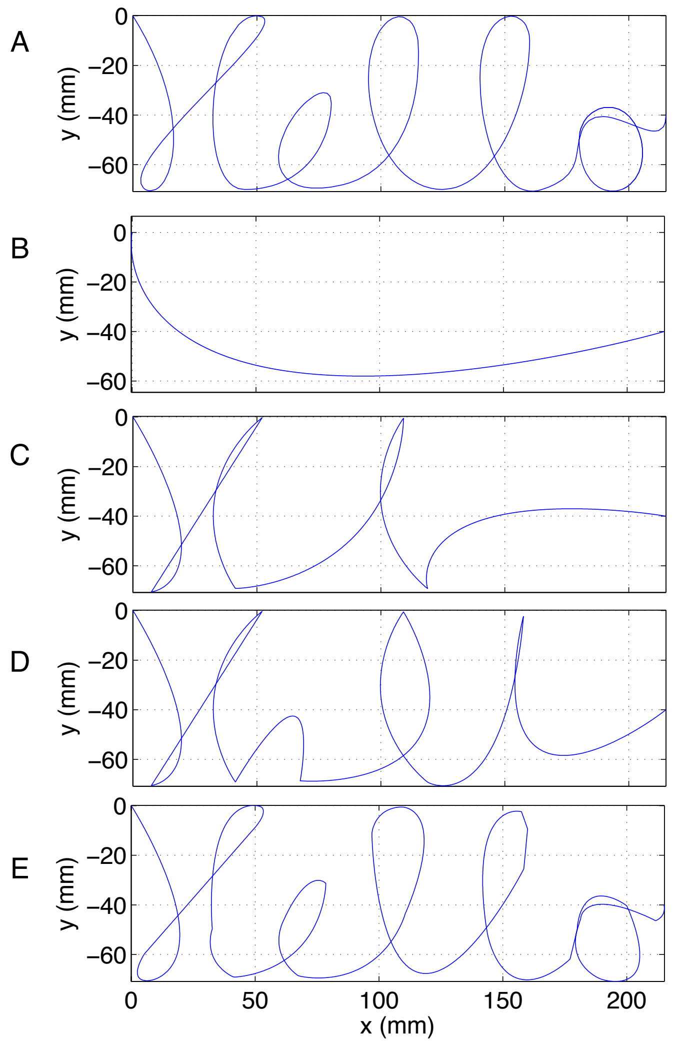


Fig. 7.8: Decomposition of the word "Hello". (A) The demonstrated trajectory. (B) First iteration: the complete trajectory is approximated by one single stroke, i.e. one primitive chosen in the MJM-51 set. (C) Iteration 7: the algorithm has identified main features in the demonstration. (D) Iteration 9: at each further step, more features are captured and represented. (E) The final decomposition and representation.

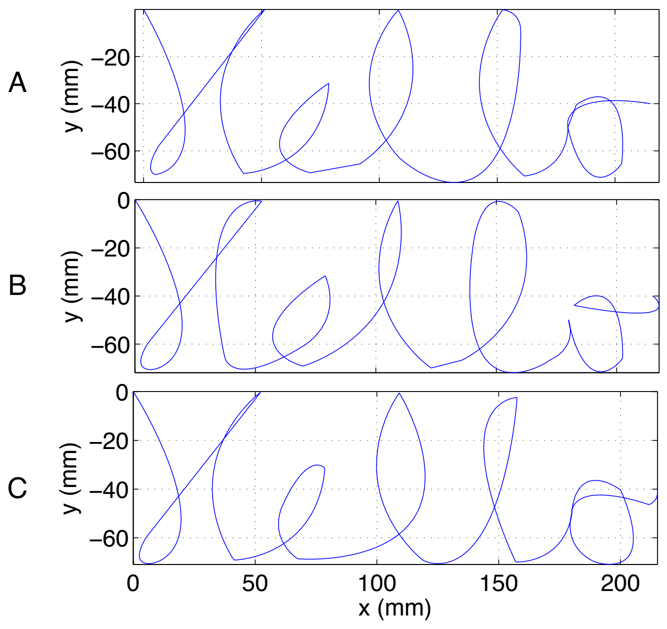


Fig. 7.9: Decomposition of the word "Hello" with different settings. (A) Final decomposition with Eq. (7.3) as matching criterion (area between trajectories). (B) Final decomposition with Eq. (7.1) (maximum error). (C) Final decomposition with Eq. (7.2) and precision $p = 20$. The decompositions are slightly different in each case, however, the capability of decomposing and representing the demonstration appear robust with respect to different matching criteria and precision of primitives.

primitive set. In this case, the approximation appears less accurate and straight lines are frequently used. However, the main features of the demonstration are captured indicating that the smaller set of primitives resulted in a more abstract representation. It is interesting to note that the decomposition with the larger primitive set results in better approximation with fewer parts. Although this fact appears intuitive, these experiments show that the current method achieves this trade-off that emerges autonomously when the movement primitive library changes. It can be concluded that the proposed method adapts autonomously to exploit the specific primitives, i.e. the available skills, of the agent or robotic platform that performs the movements.

The reconstruction capabilities, already proven earlier with the test in Fig. 7.5, are preserved also when decomposing and reconstructing longer trajectories. A decomposition was run on the dataset in Fig. 7.10A, corrupted by the addition of $\pm 1\%$ noise to each sampling point. The decomposition proceeds on this noisy dataset similarly to the case without noise. Fig. 7.11 shows that the reproduced trajectory is an interpretation of the noisy data. This simulation proves that the proposed algorithm employs its generalization capabilities to filter noise and detect relevant features in the demonstrated trajectory.

7.4 Discussion

In the proposed algorithm the idea is to decompose an arbitrarily complex trajectory using the agent's pre-defined primitives during an iterative process of learning-by-doing. The process starts with a rough approximation of the demonstrated trajectory and approximates step by step the features of the input data by a progressive decomposition. Segmentation points are discovered simply by a criterion of maximum error between demonstration and reproduction. Such a trivial criterion that focuses only on geometrical features proved nevertheless surprisingly effective and robust. The final result is a sequence of primitives that is in effect an intelligent reading of a demonstrated trajectory represented as a general and abstract concept. The strength of the algorithm lies in the primitive-centered and progressive search, which uses existing skills and implicitly solves data-induced problems like noise and discontinuous derivatives.

Finding segmentation points and fitting sub-trajectories is potentially an intractable problem if considered exhaustively. The proposed method suggests candidate segmentation points taking advantage of progressive approximations. The computation required to generate a reproduction increases with the number of iterations and the number of available primitives. The removal of constraints, i.e. the search of primitives that bypass segmentation points, is done at a the computational cost of matching the locally segmented demonstration with primitives. However, removing segmentation points results in more general solutions, which justify the additional computation. The removal of constraints is effectively a search procedure to avoid local minima in a highly dimensional search landscape.

For simplicity, the current study considers finite movement primitive libraries in which each primitive has a fixed geometry. An alternative approach consists in using primitives with variable geometry that use one parameter to change certain features as, for example, the curvature. The use an infinite-set of primitives requires a different representation, but does not increase the computational complexity of the search. In fact, a larger variety of geometries can be implemented with fewer tune-able movement primitives. The extension of the algorithm to infinite-set of primitives is promising particularly in the cases where high precision and compact representations are required.

The algorithm appears to have generalization capabilities even if it decomposes trajectories from one single demonstration. The generalization capability, noticeable particularly in Fig. 7.5, derives from the interpretation of the demonstration according to the agent's movement primitive library, and less emphasis on the original data. The reconstruction from noisy data in particular shows the generalization capability in reconstructing straight lines, identify correct curvatures, as well as maintaining cuspids, as clearly shown in Fig. 7.11.

The criteria upon which the algorithm is constructed (Sec. 7.2) represents the *intelligence* of the decomposition, which is intended to mimic loosely human processes of understanding, acquiring and reproducing articulated movement or trajectories. For this reason, the proposed algorithm focuses less on the input data itself and more on the quality of the procedure applied to interpret it. The use of move-

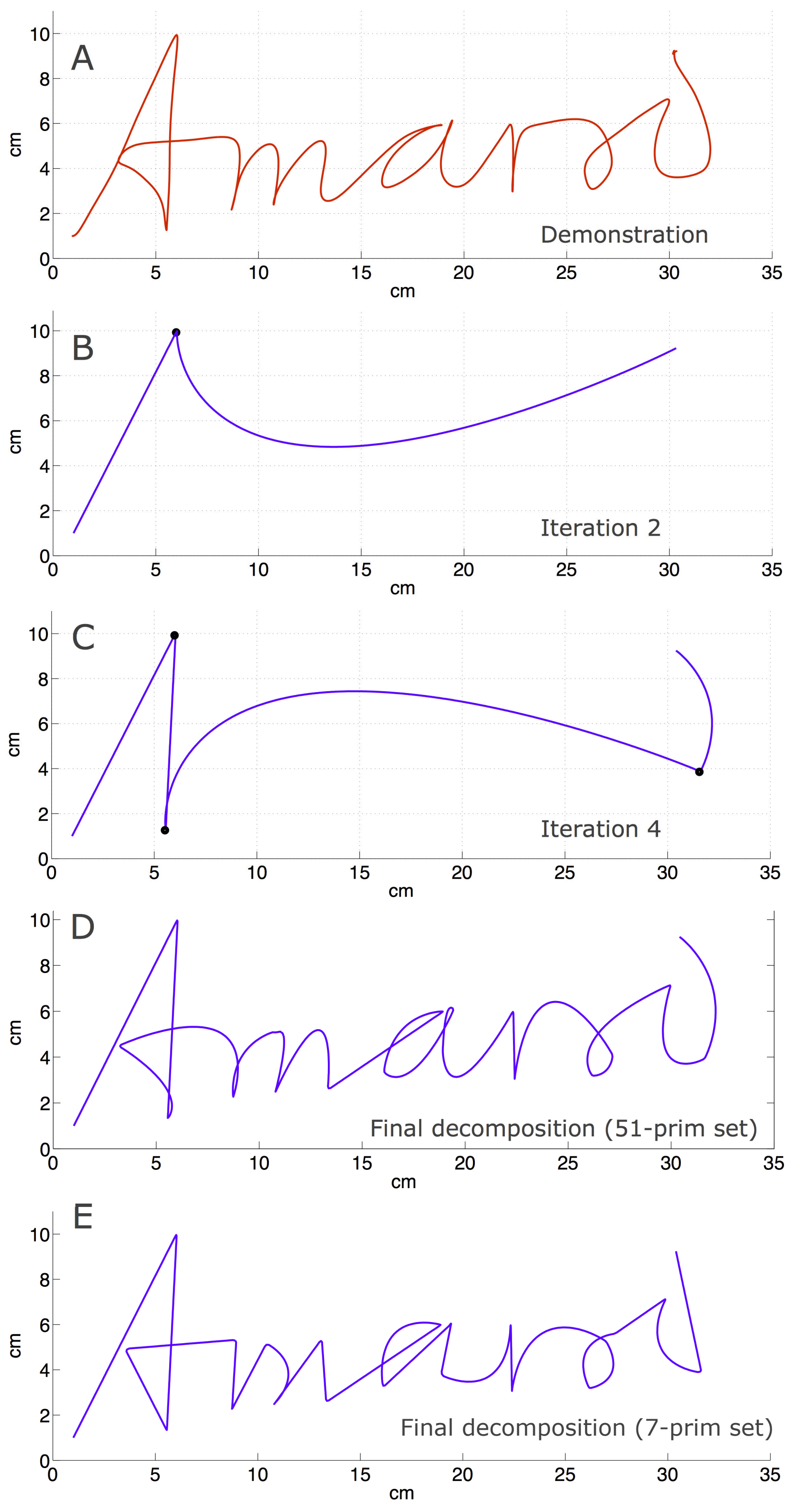


Fig. 7.10: Decomposition of the hand-written trajectory “Amarsi”. (A) The demonstrated trajectory recorded from handwriting. (B) Iteration 2 during the composition with the large 51-primitive set in Fig. 7.1B. (B) Iteration 4 during the decomposition. (C) The final decomposition and approximation using the set of 51 primitives. The demonstration was decomposed in 18 parts (D). The final decomposition and approximation using the set of 7 primitives in Fig. 7.1A. The algorithm decomposed the trajectory in 21 parts. Although the accuracy with the 7-primitive set is lower than with the larger set, the reproduction appears readable and to some extent a more abstract representation of the demonstration.

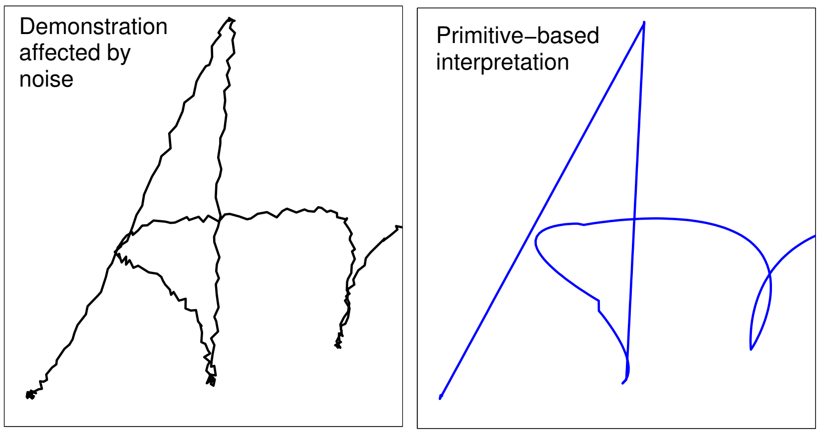


Fig. 7.11: Detail of the decomposition of a noisy version of the hand-written trajectory “Amarsi”. (Left) Noisy demonstration. (Right) Reconstruction using the 51-primitive set.

ment primitives implies inevitably the classification of imprecise and noise-affected demonstration into well defined trajectories. Therefore, such a process causes the loss of accuracy from the demonstrated data. However, such an accuracy may not be descriptive of features of the demonstration. Abstract representations are more compatible with hypotheses on how humans and animals represent and execute movements.

The precision parameter, encoding the second dimension or thickness of a movement primitive, determines effectively to which level small details in the demonstration need to be reproduced. As a consequence, high precision means that a noisy demonstration is reproduced accurately down to small details, while low precision means that the trajectory is more heavily interpreted according to the movement primitive library. It is important to note that a low precision parameter is not equivalent to high noise filtering. In fact, cuspids and prominent features of the demonstrations are nevertheless captured as indicated in Fig. 7.11.

The method is tested by using one demonstration only for each trajectory. A promising extension is to use multiple demonstrations of the same trajectory to increase the generalization properties of the algorithm. In particular, more observations of one demonstration are likely to have variations but retain relevant features. One extension is to increase its capability of generalizing trajectories by finding one decomposition of a set of similar demonstrations.

The algorithm uses primitives and demonstration in a two dimensional space. The method can be extended and applied to a 3D scenario because primitives and matching functions can be equally generated and computed in 3D space. It is conceivable that primitives in a 3D space may nevertheless used in a piece-wise planar fashion [Sternad and Schaal, 1999], and that truly 3D trajectories like a helix are relatively rare. The extension presents challenges but is a promising venue for reproducing fully-fledged robotic movements in space.

The trajectories considered in this study were only determined by the geometry without velocity profiles. In effect, releasing the constraints on velocity allows agents to reproduce complex demonstrations by freely choosing from their own primitive library with given velocities representing their own capabilities. Extensions of the algorithm could include also the velocity profiles. The addition of kinematics may imply that velocity cannot drop to zero at segmentation points, introducing strict constraints to the search. In effect, whereas kinematics are essential in dynamics movements such as walking, they become less stringent in object manipulation and marginal in drawing and writing. As the respect of kinematics constraints depends heavily on the precise field of application, tailored algorithms may be required.

The proposed method focuses on the decomposition of trajectories and does not consider the learning of new movement primitives. The results in this chapter indicate that the movement primitive library is important to achieve particular required performance, and in particular is crucial in interpreting noisy or corrupted data. It is natural to ask how the algorithm can be adapted to extend the available set of primitives while decomposing. A promising research direction is that of integrating the current method in a more powerful algorithm that learns additional primitives with experience, which is discussed in depth in Chapter 8. Additionally, certain sequences of primitives that repeat themselves frequently could be assimilated as a new longer primitive as discussed in Chapter 6, thereby accelerating the search in future occurrences of the given sequence.

7.5 Conclusion

A new approach to decompose and reconstruct complex trajectories by means of a movement primitive library is proposed. The method starts decomposing a complex trajectory with one initial single primitive and progressively increases the accuracy of the approximation through an iterative process. This approach allows an initial reduction of the search space with the identification of prominent features of a demonstrated trajectory. Subsequently, the iterative search makes use of newly found segmentation points to search locally better solutions and escape local optima. The agent-centered process offers a new way of interpreting data as a function of the agent's skills, which may represent various optimal primitives generated with established methods. The algorithm proves robust and displays remarkable generalization and feature extraction capabilities. In particular, the algorithm is suited for reconstructing trajectories from corrupted and noisy data. Diverse robotic platforms with different degrees of accuracy and motor patterns could benefit from this method while learning progressively and autonomously the decomposition of complex trajectories. Promising extensions of the algorithm include the applications to a variety of tasks such as imitation learning, learning of complex motion patterns, gestures, object manipulation, software-based and robotic handwriting.

CHAPTER 8

BOOTSTRAPPING OF MOVEMENT PRIMITIVES

The concepts discussed until now described issues of how to represent motion primitives in Chapter 3 and Chapter 4, how to combine them to one complex trajectory in Chapter 5 and how to refine existing movement primitives in Chapter 6. In Chapter 7 a decomposition approach is described that are suitable for segmenting complex trajectories with a movement primitive library. Based on these concepts one motion skill architecture is designed in this chapter. It learns starting from only one basis straight movement primitive and creates new movement primitives which will be refined with further training. This new modular framework maintains a suitable movement primitive library which is used to perceive known movement primitives in a complex trajectory. It creates new movement primitives autonomously and refines existing movement primitives with a semi-supervised learning approach. The learning behavior of this learning cycle is evaluated in one toy example and real world applications. Additionally, the learning cycle is applied in a full skill learning architecture, which is used to teach the humanoid robot iCub. The work described in this chapter is also published in [Lemme et al., 2014b].

8.1 Learning cycle of movement primitives

Single motion primitives are typically small entities, which exert their whole motion generation power only in combination with several. This behavior needs to be coordinated in a larger learning architecture, which can trigger each primitive in context of the given task. A prerequisite to coordinate this stream, is a connection between the perception and actions space. The actions are then given by the set of primitives and the perception is able to identify possible primitives in the task. However, the main question in such an learning architecture is: what kind of primitives are needed in the task?

The issue of identifying and extracting movement primitives autonomously from complex trajectories using a MPL has been addressed in [Meier et al., 2011, 2012; Grave and Behnke, 2012]. The decomposition is accomplished by maximizing the likelihood that the demonstrated trajectory can be approximated by a sequence of movement primitives from the library. If no primitive exceeds a given threshold for the likelihood, a new MP is learned from the demonstration. In [Lee and Ott, 2010], an incremental learning scheme is proposed to sequentially learn a MPL. The library is represented by hidden Markov models (HMMs), which allow recognition of the movement primitives from an on-line stream of control commands. Another stochastic learning approach used in an incremental learning scenario is introduced in [Kulic et al., 2008b, 2009, 2011]. HMMs are used to represent movement primitives and also to obtain stochastic segmentations of complex motions into known primitives. In all these mentioned approaches segments of the demonstration which are not known in the MPL are used to learn new movement primitives as proposed in the programming by demonstration paradigm. In other approaches, segmentation points are solely defined by geometrical or kinematic features, e.g. critical points [Mo-

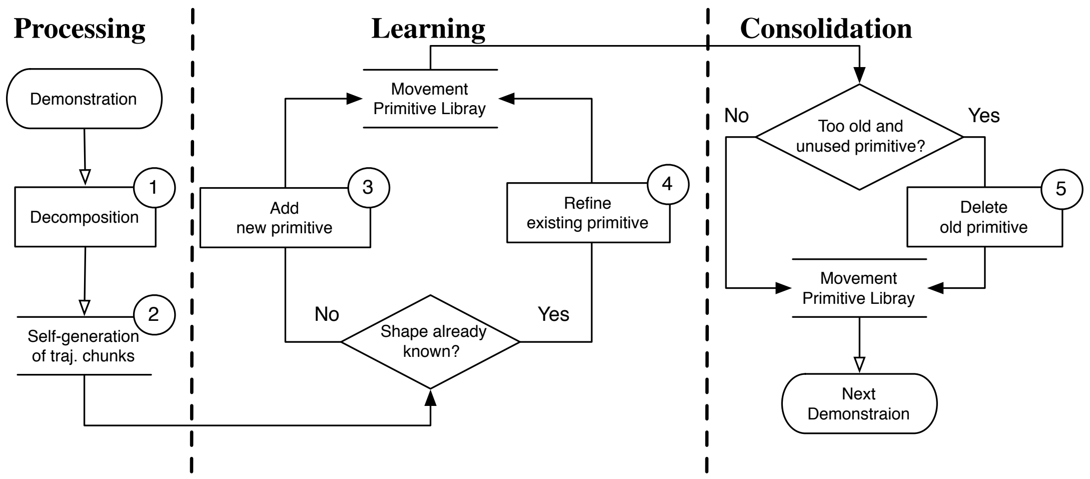


Fig. 8.1: Flow chart describing the bootstrapping cycle of the library. First is the decomposition (1) of the observed complex trajectories. The result is used to self-generate trajectory chunks (2) for the next learning step. Learning consists of two operations, either adding (3) or refining (4) movement primitives. At last the library is consolidated by deleting (5) old and unused movement primitives.

han et al., 2010], zero velocity [Reinhart et al., 2012], spatio-temporal features [Jenkins and Mataric, 2002], or general motion laws believed to underly human motion generation, e.g. two-thirds power law [Endres et al., 2013], minimum jerk [Rohrer and Hogan, 2003]. These methods can be characterized as approaches where predesigned motion features directly determine the structure of the MPL.

A general approach to solve this issue is, by using a decomposition algorithm which identifies invariant features in the perceived trajectory and implement these features in a primitive representation. The downside of this approach is that one is imitating the seen demonstration, which may is beyond the capabilities of the robot motion apparatus. An alternative approach provides the reinforcement learning community, where optimization methods are found which can be used to change the features of the learned primitive to be more suitable for both the robot and the task.

In this chapter a open-ended learning paradigm is proposed using an autonomous learning architecture, where primitives are again represented as dynamical systems and organized in a library structure. This library of motion primitives is used to compose complex motions which can represent the task as a sequence of motion primitives. The main idea is that the available primitives are used to represent every task demonstrated by the tutor with its own motion capabilities and evolve further by repeating the task multiple times.

A MPL is bootstrapped by using the motion capabilities currently available, starting from only one straight MP embedded into the propose autonomous learning architecture. The MPL is used to compose and decompose complex trajectories by sequencing primitives from the library. The main contribution is to bootstrap new movement primitives by concatenating frequently used movement primitives to more complex ones. This is done by using self-generated training data, based on the notion of co-articulation [Sosnik et al., 2004] found in the human motion training process and is used also in Chapter 6 to develop a learning method.

That is, demonstrations do not serve as immediate training targets. Instead, reproduced trajectories generated with the current MPL are used for semi-supervised learning of new movement primitives. The MPL is further refined during multiple repetitions of a task without the prior selection of specific motion features. The concept of semi-supervised learning or self-learning is mainly known in the classification and object detection domain [Nigam and Ghani, 2000; Rosenberg et al., 2005], where e.g. a classifier is trained at first with a small training data set. Then this classifier labels new data samples, which are used again to train the same classifier.

8.2 Semi-supervised bootstrapping of movement primitives

In this proposed learning architecture the concepts introduced during the last chapters of this thesis are applied. The core idea of the proposed approach is to combine three major steps. The first step is the *processing*, where the existing MPL is used to decompose a complex trajectory and generate new training data for the MP learning. The second step comprises *learning*, where the new training data is organized in training data sets for creating new or refine existing movement primitives. The third step *consolidate*s the MPL by deciding which MP needs to be deleted.

Fig. 8.1 illustrates the steps of one such bootstrapping cycle obtained from decomposition. The sequence of movement primitives (see Fig. 8.1(1)) is used to self-generate articulated trajectory chunks (see Fig. 8.1(2)) as is described in more detail in Sec. 8.2.1. These chunks are used either to create new movement primitives (Fig. 8.1(3)) or to refine existing movement primitives (Fig. 8.1(4)) as described in detail in Sec. 8.2.2. The MPL holds additional information for each MP, e.g. how frequently a MP was used for decomposition and for how long it has been part of the library. After the *learning* phase, unused primitives are deleted (Fig. 8.1(5)). Due to the autonomous decomposition, it is not necessary to involve a human designer or teacher, who organizes training data before learning.

Movement primitives are represented by the alternative DMP approach using the ELM to learn the correction forces, as introduced in Chapter 5.

8.2.1 Processing of complex trajectories

The processing step tackles two tasks. First is the decomposition of the complex trajectory w.r.t. the known MPL (see Fig. 8.1(1)). Second is the preparation of training data for the learning step (see Fig. 8.1(2)).

Decomposition with respect to a known movement primitive library

In order to decompose arbitrarily complex 2D trajectories w.r.t a given MPL the decomposition algorithm as described in Chapter 7 is applied. The algorithm considers a MPL where movement primitives are represented by a collection of discrete shapes. This allows to use any MP representation.

The iterative process of the decomposition starts with a rough approximation of the demonstrated trajectory by a single MP and approximates step-by-step the geometric features of the trajectory. Segmentation points are discovered simply by a heuristic of maximum error by comparing the demonstration with the current best approximation.

Self-generating training data

The bootstrapping cycle starts with only one single MP in the MPL, which is a straight line generated by the spring-damper system Eq. (5.1) with constant $f = 0$. The self-generating of new training data is based on the concept of co-articulation illustrated in Fig. 8.2:

a) The process starts with an initial decomposition of a given demonstration (see Fig. 8.2(a)).

b) Each used primitive (see Fig. 8.2(b) straight lines) and two consecutive primitives from the *reproduced* trajectory (see Fig. 8.2(b)(1,2)) are used.

c) and add these extracted trajectories (compare Fig. 8.2(c)) to the training data (see Fig. 8.1(2)).

If a number of segments J are used for the decomposition, then $N = 2J - 1$ new trajectory chunks $\Omega_{\text{chunk}} = \{\omega^1_{\text{chunk}}, \dots, \omega^N_{\text{chunk}}\}$ are sent to the *learning* step as illustrated in Fig. 8.1 and described in the following.

8.2.2 Semi-supervised learning

The *processing* step described in the previous section delivers chunks of trajectories Ω_{chunk} as shown in Fig. 8.2 which are used as training data. Each MP in the MPL is represented by a normalized trajectory $\Omega_{\text{MP}} = \{\omega^1_{\text{MP}}, \dots, \omega^M_{\text{MP}}\}$, where M is the number of the currently available movement primitives. The normalization keeps the length ratio ratio and the angle between the articulated movement primitives. To determine if a MP is already in the MPL, each trajectory in Ω_{chunk} is compared to each MP in the Ω_{MP}. The focus in the comparison is on the geometrical shape called "path" of the trajectory. Therefore, each trajectory $\omega_* \in \Omega_{\text{MP}}, \Omega_{\text{chunk}}$ is resampled with T_{norm} equidistant points representing only the path $\hat{x}_*$ of the trajectory ω_*. The resampling is done by cubic spline interpolation. Two paths $\hat{x}_{chunk}$ and $\hat{x}_{MP}$ are compared by the R^2 metric:

$$R^2 = 1 - \frac{\sum_i \left(\hat{x}_{\text{MP}}(i) - \hat{x}_{\text{chunk}}(i)\right)^2}{\sum_i \left(\hat{x}_{\text{MP}}(i) - \bar{\hat{x}}_{\text{MP}}\right)}, \tag{8.1}$$

where $\hat{x}_{\text{MP}}$ is the path represented by a MP stored in the MPL and $\hat{x}_{\text{chunk}}$ is the new path from the training data. A perfect match is indicated by $R^2 = 1$.

With this similarity measure the training data can be organized and assign each trajectory $\omega_{\text{chunk}} \in \Omega_{\text{chunk}}$ to the training data set corresponding to the MP represented by ω_{MP}. A similarity threshold Θ, which is chosen between $0 \ll \Theta < 1$ is used to determine if ω_{chunk} and ω_{MP} are similar. Now two different cases can be identified: (A) no $\hat{x}_{\text{MP}}$ is similar to the observed $\hat{x}_{\text{chunk}}$. Then, $\hat{x}_{\text{chunk}}$ becomes a $\hat{x}_{\text{MP}}$ (i.e. a representative trajectory in the MPL for a DMP which does not exist yet) and ω_{chunk} is added to the corresponding training data set as the first trajectory. Note that the number of MP in Ω_{MP} is growing in this case, although no DMP representation is learned for this specific MP. However, by using this schema all follow up trajectories in Ω_{chunk} are compared also to this representative of the new MP. (B) a DMP representation already exists with a similar path $\hat{x}_{\text{MP}}$. Then the trajectory x_{chunk} is added to the training data set of the corresponding MP.

After the organization of the training data into training data sets the learning is triggered. To cope with the two possible cases (A) and (B), the on-line variant of the ELM denoted by on-line sequential ELM (OS-ELM) [Liang et al., 2006] is applied. Learning is organized in an initial learning phase and a sequential learning phase as explained in the following.

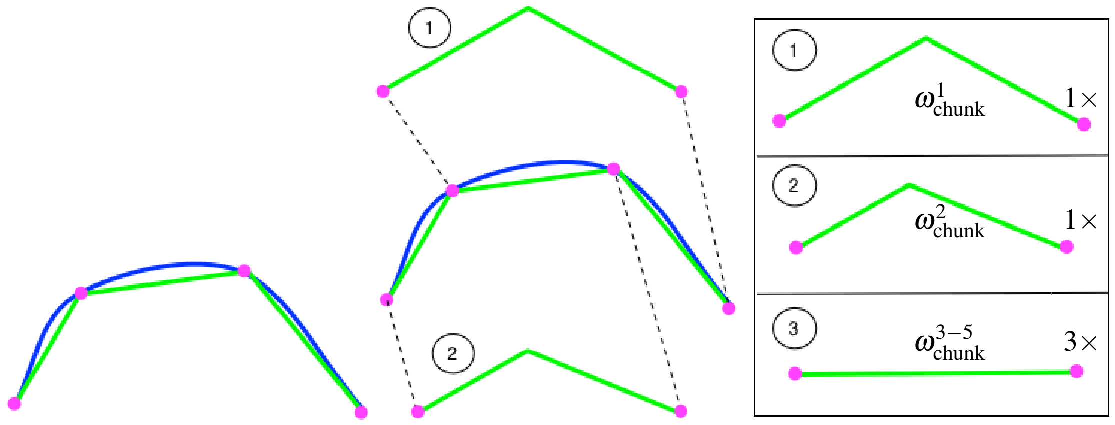

1
2
1
ω^1_{chunk}
1×
2
ω^2_{chunk}
1×
3
$\omega^{3-5}_{\text{chunk}}$
3×
(a) Initial decomp. (b) Co-articulation (c) New training data

Fig. 8.2: Toy example which illustrates the self-generation of new training data. The initial decomposition is shown in Fig. 8.2(a), where the segmentation points are indicated in magenta points and the reproduced trajectory produced by the primitives in green straight lines (i.e. consider that the library consists only of one straight line MP initially). This sequence of segments are then used to create new MP candidates by articulating two consecutive segments as shown in Fig. 8.2(b). These trajectory chunks are normalized and used as training data for learning. Fig. 8.2(c) illustrates the normalized training data.

(A) the training data set can contain one or more trajectories and is used in the initial learning phase of a new DMP representation with $k = 0$:

$$\mathbf{P}_k = (\mathbf{H}_k^T \mathbf{H}_k + \lambda \mathbb{1})^{-1}, \tag{8.2}$$

$$\mathbf{W}_k^{out} = \mathbf{P}_k \mathbf{H}_k^T \mathbf{Y}_k, \tag{8.3}$$

where $\mathbb{1}$ is the identity and $\lambda > 0$ a regularization parameter. The matrix $\mathbf{H}_k = (\mathbf{h}(s(1)), \dots, \mathbf{h}(s(N_{\text{tr}})))$ harvests the hidden states for each input $s(l) : l = 1 \dots N_{\text{tr}}$ in the training data set.

(B) A DMP representation already exists and needs to be refined with the new training data set:

$$\mathbf{P}_{k+1} = \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbb{1} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k, \tag{8.4}$$

$$\mathbf{W}_{k+1}^{out} = \mathbf{W}_k^{out} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{Y}_{k+1} - \mathbf{H}_{k+1} \mathbf{Y}_k). \tag{8.5}$$

8.2.3 Consolidation

After the *learning* step, the library is searched for primitives which have not been used for the decomposition. To this aim each MP receives a life counter γ when it is created. The counter decreases if this primitive is not used. Otherwise it is incremented. If $\gamma < 1$, then the MP gets deleted from the library. The straight MP is an exception and is not deleted. A large γ results in a big number of movement primitives especially in the first iterations of the bootstrapping cycle. If γ is set too small the bootstrapping of primitives is limited. In general this parameter is necessary to avoid a over-complete library, which would result in a slower bootstrapping cycle.

8.3 Learning process of the bootstrapping cycle from 2D trajectories

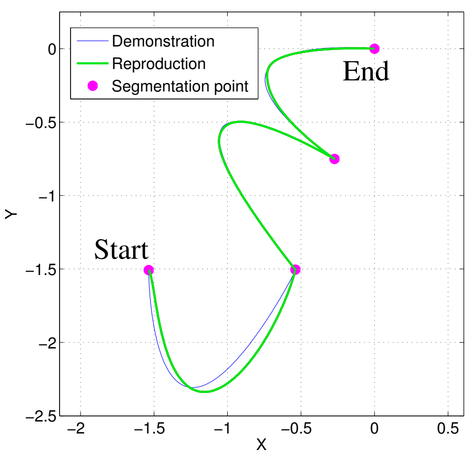
In this section, the general learning process of the proposed bootstrapping cycle is evaluated. In Tab. 8.1, an overview of the necessary free parameters is given, which need to be specified before starting the bootstrapping cycle and also how they are set for the following experiments.

8.3.1 Proof of concept

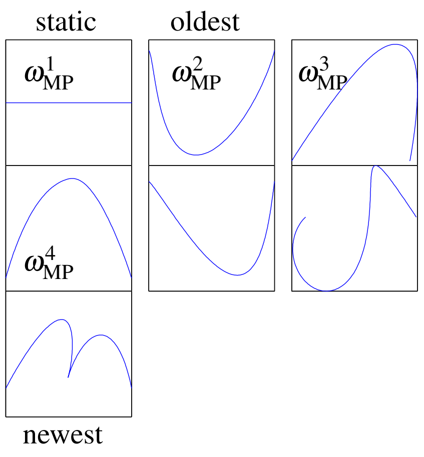
As a first example to display the learning behavior of the new approach, a trajectory in form of three concatenated parabola shapes with different scaling and orientation is chosen as illustrated in Fig. 8.3.

	Experiment reference	Sec. 8.3	Sec. 8.4
	Decomposition		
ε	Error margin	0.025	0.075
δ	Challenge seg. points	2	
	Extreme learning machine		
λ	Regression parameter	$1e-10$	
R	Number of hidden neurons	15	
$\mathbf{w}^{inp}, \mathbf{b}$	Input weights & biases	uniform in $[-1, 1]$	
	Dynamic Movement Primitive		
K	Stiffness parameter	200	
D	Damping parameter	$2\sqrt{K}$	
τ	Time-steps	200	
	Clustering		
θ	Threshold to add new movement primitives	0.98	
γ	Threshold to delete old movement primitives	1	

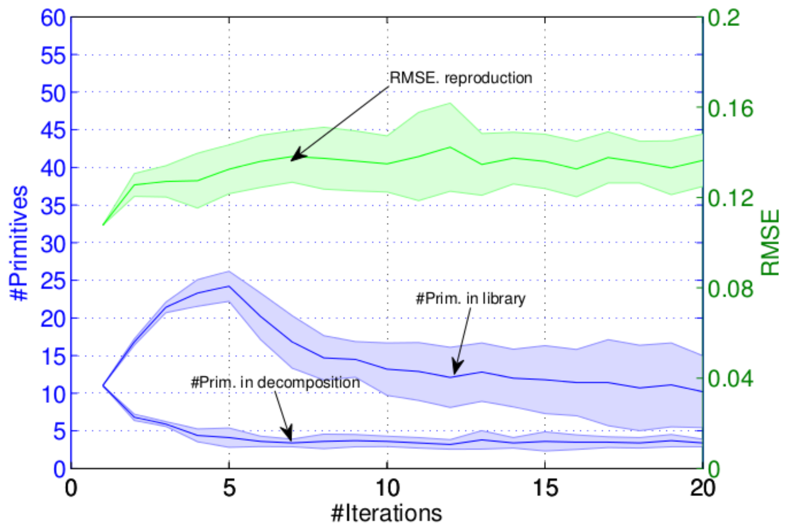
Tab. 8.1: Important parameters which need to be set by the user.



(a) Reproduction



(b) Library

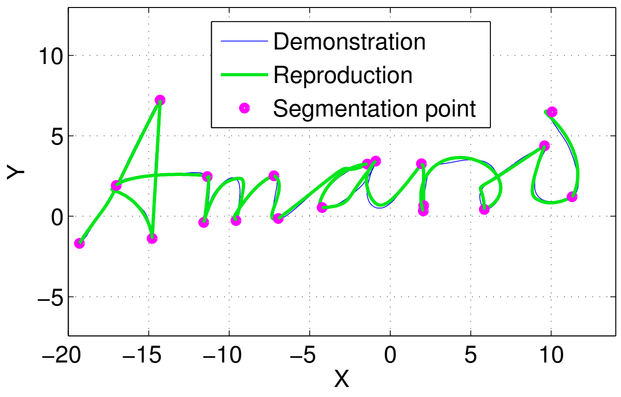


(c) Evaluation

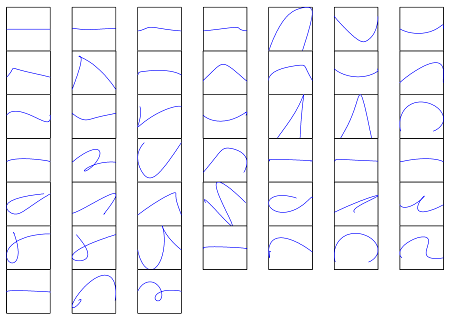
Fig. 8.3: A toy example to illustrate the learning process. The decomposition after 20 iterations of the learning cycle is shown in Fig. 8.3(a). The learned library Ω_{MP} is shown in Fig. 8.3(b). The movement primitives ω^2_{MP},ω^3_{MP} and ω^4_{MP} are used in the 20th iteration. The number of used primitives in the decomposition during the learning process and the number of primitives in the MPL is shown in Fig. 8.3(c) together with the approximation error (RMSE), over 10 trails and 20 iterations.

The learning cycle for executed for $G = 20$ iterations. The result is shown in Fig. 8.3. The approximation of the complex trajectory is shown in Fig. 8.3(a) and the learned library is shown in Fig. 8.3(b). It contains seven different shapes, beginning with the straight MP in the upper left corner. Note that besides the straight line and the movement primitives used in Fig. 8.3(a), three more primitives are in the MPL. Those additional primitives can be categorized into two different groups. First, there are old primitives which have not yet been deleted. Second, there are new movement primitives which have been added in the last iteration of the bootstrapping cycle, e.g. the bottom MP shaped like the letter ’m’.

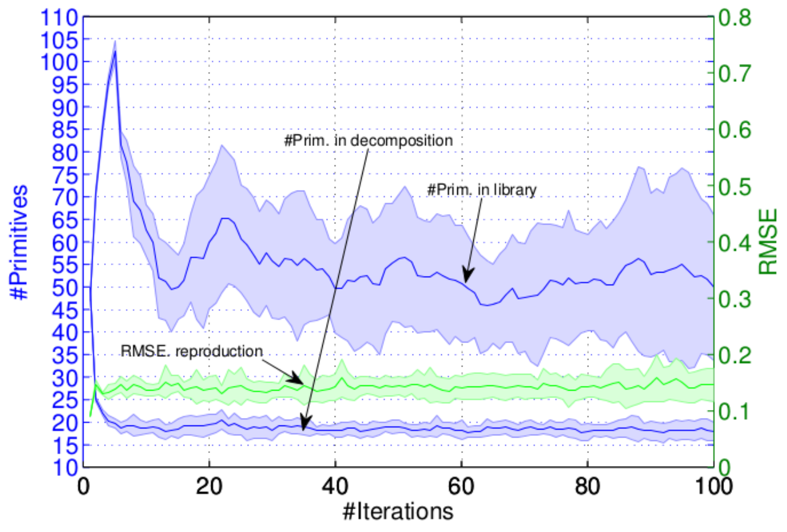
The number of primitives is evaluated during 10 trials of the bootstrapping cycle. In Fig. 8.3(c), the used number of primitives over time is illustrated. Note that in the beginning of the learning process the total number of movement primitives in the library is growing quickly. After five iterations, however, the number of movement primitives starts decreasing due to the deletion of unused primitives. The number of primitives, which have $1 \ll \gamma$ and are available for the decomposition is also shown in Fig. 8.3(c).



(a) Reproduction



(b) Library



(c) Evaluation

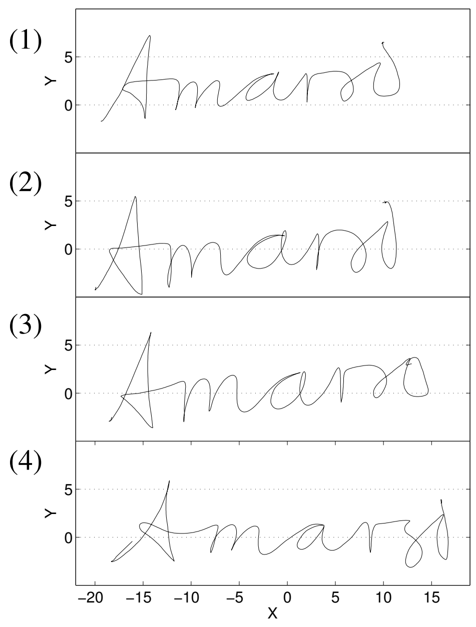
Fig. 8.4: A recorded handwritten demonstration is used for bootstrapping a MPL. The decomposition after 100 iterations of the bootstrapping cycle and 10 trails is shown in Fig. 8.4(a). In Fig. 8.4(b), all movement primitives in the MPL after 100 iterations are shown. Note that the bootstrapping found complex primitives representing complex shapes like an ’s’, ’n’ or the first part of an ’a’. The number of used primitives in the decomposition during the learning process and the number of primitives in the library are shown in Fig. 8.4(c) together with the approximation error (RMSE).

After the fifth iteration, the number of used primitives is almost constant and varies only marginally, which means that the bootstrapping is finished.

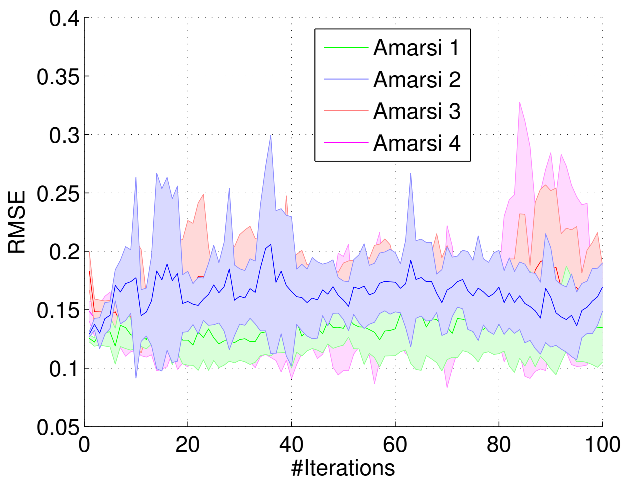
8.3.2 Learning from handwriting

After the proof of concept, a more complex trajectories from human handwriting are considered. A human subject demonstrated four handwriting motions of the word ”Amarsi” on a tablet (see Fig. 8.5(a)). The same parameters from Sec. 8.3.1 are used (see Tab. 8.1). In Fig. 8.4, the reproduction, evaluation and the MPL after $G = 100$ iterations is shown.

In this complex scenario, the qualitative results from the proof of concept are reproduced. In Fig. 8.4(c), the characteristic increase of the number of movement primitives in the library persists longer than in the example shown in Fig. 8.3. In all trials, the number of primitives is drastically reduced before the 20th iteration. This behavior can be modified by increasing the parameter γ, which will result in a slower decay of the number of movement primitives. In comparison to the example from Sec. 8.3.1,



(a)



(b)

Fig. 8.5: Demonstrated trajectories in Fig. 8.5(a). Amarsi 1 is the used for the learning cycle and Amarsi 2-4 is used for the test set. Generalization error (RMSE) statistics for four ”Amarsi” handwriting motions in Fig. 8.5(b). Solid lines depict RMSEs averaged over 10 trials of the bootstrapping cycle. The shaded areas depict standard deviations.

the complexity of the movement primitives in the library is increased. The bootstrapping cycle based on co-articulation results in complex movement primitives, which approximate distinct shapes e.g. ’s’, ’n’ or the first part of an ’a’ (see Fig. 8.4(b)). To evaluate the performance the point wise root mean square error (RMSE) between the demonstration and the recomposed motion is used. Note that after the 20th iteration the RMSE and the number of used movement primitives in the reproduction is almost constant. However, the number of movement primitives in the library is still varying. This indicates that the bootstrapping cycle is still adding new primitives, but these are not used in the decomposition of the complex trajectory and are deleted again. The amount of variation, depends on the ability to (i) approximate the new shapes and (ii) be able to identify it again in the complex trajectory. If a perfect solution for (i) and (ii) is found, the bootstrapping cycle would end with one MP in the MPL which can approximate the full complex trajectory and the variation would be zero.

In Fig. 8.5, the generalization performance of the learned library to new demonstrations is illustrated. Learning is conducted only on one ”Amarsi” demonstration, whereas three more demonstrations serve as a test set. The test demonstrations are decomposed with the learned MPL. Fig. 8.5(b) clearly shows that the generalization to new demonstrations is possible with similar reproduction performance as on the training demonstration.

8.3.3 Static primitive set versus bootstrapped primitives

In this section, a comparison is conducted, where the approximation ability of a static and bootstrapped primitive libraries are compared. The static primitive set applied in this task is the same library of minimum jerk trajectories introduced in Chapter 7 with 51 different movement primitives. The task is to perceived the hand designed word ’Hello’ (see Fig. 7.8) and represent this complex trajectory with the respective movement primitive library. The bootstrapping algorithm learns for $G = 100$ epochs using the parameter set given in Tab. 8.1. The learning process is repeated 10 times.

In Tab. 8.2, the comparison of both approaches according to the reproduction accuracy in form of the point-wise mean square error (*PMSE*) is shown. Next to the *PMSE* the table shows the number of primitives in the library, iterations necessary in the decomposition algorithm until the representation is found and the number of used via-points.

The results indicate a very compact movement primitive representation is found by the bootstrapping algorithm. It does not only optimize the approximation ability but also reduces the time necessary in the decomposition of the complex trajectory. Note, that the number of primitives in the bootstrapped movement primitive library are also around 51 primitives. However, this is just a snapshot of the movement primitive libraries after learning for G iterations. Longer learning could change the number of stored movement primitives.

8.4 Robotic application

The bootstrapping of movement primitives is of course an important factor, however, it is not enough in a robotic scenario to actual represent a complete skill, which a robot can perform. This section gives a motion skill learning architecture and points out where the learning needs to be performed and where the learning cycle can be found in this architecture.

The bootstrapping cycle is demonstrated in a robotic scenario with the humanoid robot iCub. A human teacher demonstrates in a physical interaction with iCub (i.e. kinesthetic teach-in) how to catch a "fish" with a toy fishing rod (see Fig. 8.6). This section demonstrates the application of semi-supervised bootstrapping of the MPL in a larger motion skill architecture.

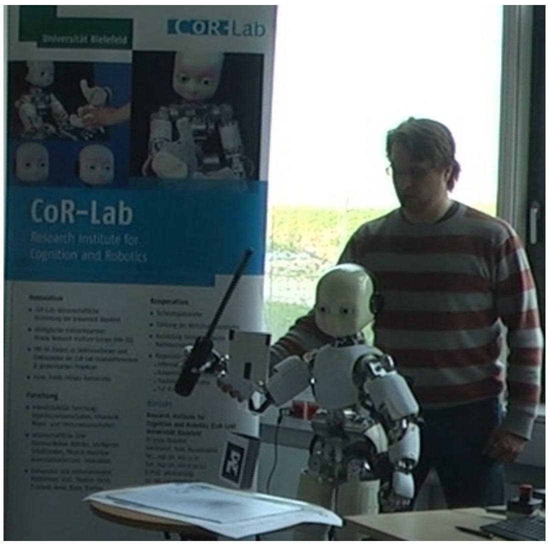


Fig. 8.6: Kinesthetic teaching of the fishing motion skill with iCub. iCub holds a fishing rod in its right hand and tracks the marker at the fishing hook by moving his head. The fish is indicated with a marker for visual tracking. The fishing hook is also static in orientation.

8.4.1 iCub Learns How to Fish

The MPL is bootstrapped from complete demonstrations of a skill in an unsupervised fashion without predesigned or manual segmentation. Both the tool kinematics as well as the MPL can be re-learned and refined over the course of repeated demonstrations, i.e. are subject to open-ended learning which is also reflected on an architectural level in terms of a changing set of movement primitives.

In Fig. 8.7, the experimental setup for learning the fishing skill is illustrated. To indirectly be able to track the fishing hook, the robot is equipped with a 3D marker tracker in order to perceive the position of the fishing hook slightly below the marker. iCub cameras located in the head and the Image Component Library (ICL) available from here [Elbrechter et al., 2013] are used for 3D tracking of the marker.

Comparisons				
	PMSE	#Primitives	#Iterations	#Via-points
static set	6.9984	51	25	21
learned set	3.65 ± 1.92	52.10 ± 11.59	16.9 ± 1.66	14.2 ± 1.4

Tab. 8.2: Comparison between the representation of complex motions using a fixed set of primitives and the bootstrapped set of primitives is presented. The bootstrapped set of primitives is achieved after 100 iterations and the bootstrapping process is repeated 10 times.

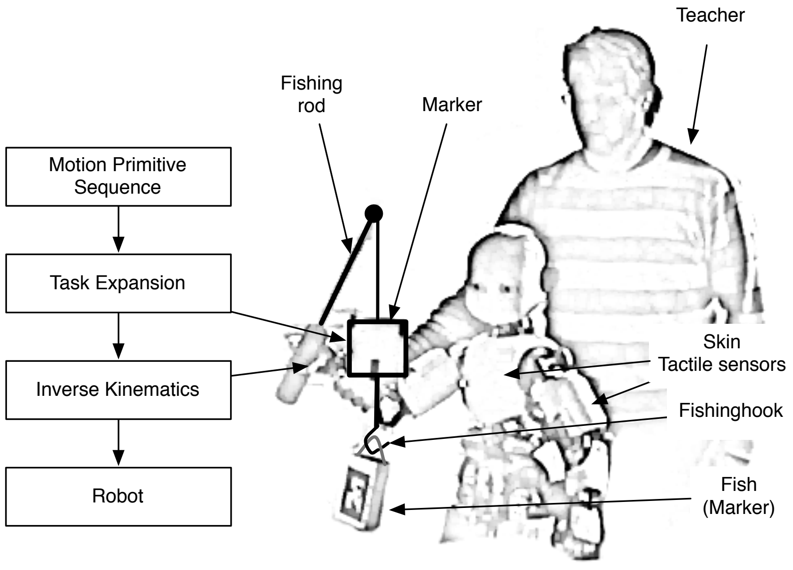


Fig. 8.7: Robotic setup. The teacher guides the right arm of iCub, such that the fishing hook catches the fish. Thereby, iCub tracks the marker attached at the fishing rod by means of head motions. Tactile sensors help in the human-robot interaction by (de-)activating the impedance control mode.

Note, however, that the robot is not able to control the tool in the beginning, i.e. to position the fishing hook at a target, nor does it have a representation of a complex motion which is necessary to hook up the fish successfully. Both issues of learning the tool kinematics and the formation of a compact MP representation of the tool-tip motion are addressed in the skill architecture.

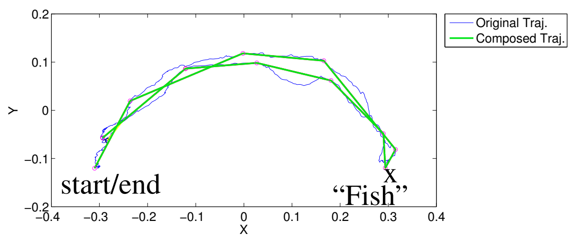
The following representational levels are shown in Fig. 8.7:

- *Movement Primitive Sequence:* Motions of the fishing hook are encoded as a sequence of movement primitives, which are formed previously by the proposed bootstrapping cycle. The motions are defined in a two-dimensional space which is sufficient to explain the considered fishing motions. Elements of this space are denoted by u and describe the projected 3D position of the marker attached to the fishing hook. The projection is performed by a principal component analysis (PCA).

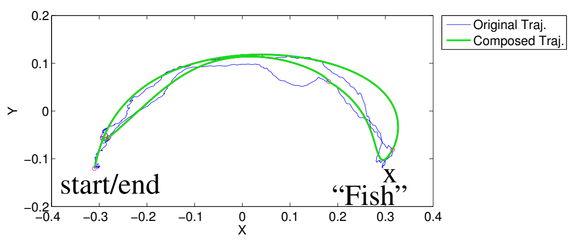
- *Task Expansion:* The task expansion maps the current target from the compact movement space u to an explicit task formulation, i.e. end effector position and orientation of the right arm. For this purpose, the 2D target u generated by the current MP is first projected back to a 3D position by the PCA. Then, this 3D position of the fishing hook is mapped to the end effector position and orientation of the robot arm by means of a trained feed-forward neural network. The learning of task expansions has been addressed before in [Reinhart et al., 2012].

- *Inverse Kinematics:* The inverse kinematics controller available in the iCub software repository [Pattacini et al., 2010] is used to control the right arm of the iCub according to the targets provided by the task expansion.

In the following, the formation of the MPL from a demonstration of the fishing skill is presented.



(a) Decomposition with straight lines.



(b) Decomposition with MP library.

Fig. 8.8: Decomposition of a demonstration according to [Soltoggio et al., 2012; Soltoggio and Lemme, 2013] using the bootstrapped MPL. Fig. 8.8(a) shows the initial decomposition, where the library consists only of one straight line primitive. Fig. 8.8(b) shows the accurate and compact decompositions using more sophisticated primitives.

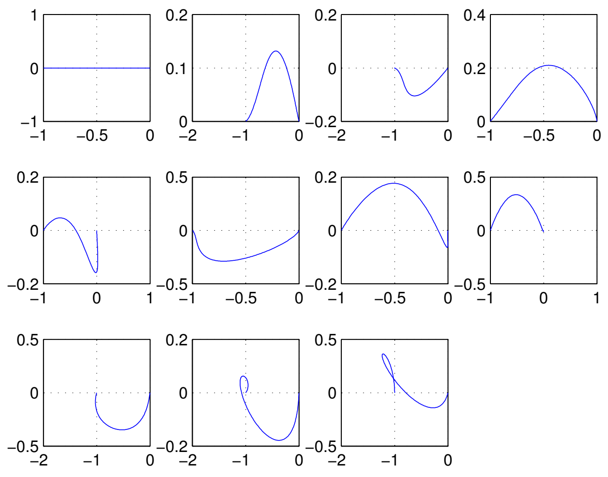


Fig. 8.9: Library of movement primitives found by the bootstrapping cycle. Note that the number and shape of the primitives is changing over time.

8.4.2 Learning how to fish

In the kinesthetic teaching phase, the human teacher can move the robot's arm while it holds the fishing rod. The controller for the right arm is switched to joint impedance mode using the force control implementation from the iCub software repository [Fumagalli et al., 2011; Parmiggiani et al., 2009], such that the arm is compliant to the external forces applied by the teacher.

After teaching, the recorded trajectories from the marker and joint angles are used for learning the tool kinematics and PCA. Then, the MPL is bootstrapped. In Fig. 8.8, the original demonstration (trajectory of the marker projected onto a plane) of the fishing skill is illustrated in blue. In Fig. 8.8(a) the initial decomposition (green) with the MPL containing only the straight line is shown. Segmentation points found by the decomposition algorithm are indicated by magenta circles. Note that almost the same parameters are used as in the previous experiments (compare Tab. 8.1) only the error margin ε is slightly increased to cope with the noise in the data.

After 4 iteration of the bootstrapping cycle, the decomposition shown in Fig. 8.8(b) is obtained. Note that the jerky demonstration is represented by smooth and well-formed movement primitives, which shows the robustness of the learning process against noise in the demonstration. The corresponding MPL is depicted in Fig. 8.9. A compact encoding of the skill is formed autonomously by the bootstrapping cycle.

The autonomous forming of a motion primitive library is demonstrated in a robotic scenario with the humanoid robot iCub. iCub is taught by a human teacher how to angle a fish with a fishing rod in a human robot interaction scenario, (kinesthetic teaching). The learning architecture to represent such a complex skill is introduced in this section. Multiple representations, which are connected to process the necessary information for each learning task is presented. The skill is represented by a set of motion primitives and the necessary information pipeline, which allows to sent suitable robot commands. The motion primitive library is bootstrapped from complete demonstrations of a skill in an unsupervised fashion starting from an initial primitive (straight line). Therefore, the number of motion primitives has not to be defined a priori. Both, the inverse tool kinematics as well as the motion primitive library can be re-learned and refined over the course of repeated demonstrations, i.e. are subject to open-ended learning which is also reflected on an architectural level in terms of a changing set of motion primitives.

8.5 Concluding remarks

In this chapter a learning architecture to bootstrap new movement primitives is introduced. The bootstrapping cycle integrates decomposition and composition of movement primitives to model complex trajectories. Movement primitives are learned semi-supervised and evolve from a single straight line primitive through co-articulation and refinement. The result is a compact movement primitive library which generalizes to novel complex trajectories. The bootstrapping of the movement primitive library addresses both segmentation and representation in a coherent framework solely based on the available movement primitives. Thus, motion features for segmentation and learning have not to be predefined.

In the current proposed architecture the decomposition algorithm is limiting the complex trajectories to be in 2D task space. However, every other decomposition algorithm can be used in this architecture as long it can use a movement primitive library to decompose the complex trajectory. Future work will address detailed evaluation of the bootstrapped movement primitives with focus on the emerging kinematic properties. Furthermore, the impact of different movement primitives representations and different decomposition methods on the bootstrapping cycle are interesting research questions.

CHAPTER 9

CONCLUSION

In this thesis, a learning architecture for bootstrapping new movement primitives from complex trajectories is proposed. Success of learning in this architecture crucially depends on the different concepts that lead to complex behavior of a robot agent. The learning cycle (depicted in Fig. 9.1) uses a movement primitive library, which is initially only one movement primitive. The movement primitive library is used to bootstrap task dependent new movement primitives. At any time in the learning process movement primitives are available, which can be used to perceive complex trajectories. The decomposed segments corresponding to primitives is used to self-generate new training data. This exploration behavior leads to new primitives or refines old ones. The semi-supervised learning of primitives is inspired by the developmental learning behavior of humans.

In Chapter 3, an extreme learning machine (ELM) approach is proposed that is suited for imitation learning tasks, denoted as Neural imprinted Vector Field (NiVF). The main contribution in this chapter is a methodology on how to incorporate stability constraints into the learning of an autonomous dynamical system such that robust motion generation becomes possible. Additionally, the accuracy of the learning is improved by providing a learning method able to approximate Lyapunov functions compliant to the training data.

In Chapter 4, a benchmark software framework is proposed, in which new methods for motion generation can be evaluated and compared to the state-of-the-art methods. The contribution is a benchmark framework with standardized evaluation criteria to evaluate the strong points and weak point of the motion generation methods which focus on accuracy criteria, but also on human-likeness. In this benchmark framework a set of realistic benchmark scenarios is provided, where robust generalization capabilities are necessary to solve the task. These benchmark scenarios are equipped with standardized perturbation parameters and a systematic way of an evaluation. This evaluation can be analyzed by the integrated tools of the benchmark software. This allows each user to compare their methods with standardized analysis and visualization. The first comparison with focus on the NiVF approach (introduced in Chapter 3) was conducted comparing seven different approaches.

After the discussion of implementing a vector field in a neural network used in a autonomous dynamical system, another movement primitive representation using a non-autonomous dynamical system is introduced in Chapter 5. An interesting application of the extreme learning machine is shown in context of the dynamic movement primitive learning paradigm. Multiple movement primitives can be learned in one global representation of the task given by the ELM. The main contribution of this network is to approach the issue of smoothly blend between shapes and to learn a sequence of primitives within one share hidden representation, which only needs a minimal control overhead.

After describing movement primitives and their representations for movement shapes and possible ways on composing these primitives, the focus in Chapter 6 changes to a refinement schema for dynamic movement primitives combined to extreme learning machines and signal depending noise. The contribution is a semi-supervised learning approach, which models a similar learning process of humans specified by the notion of co-articulation. The semi-supervised learning is applied to single and multi-

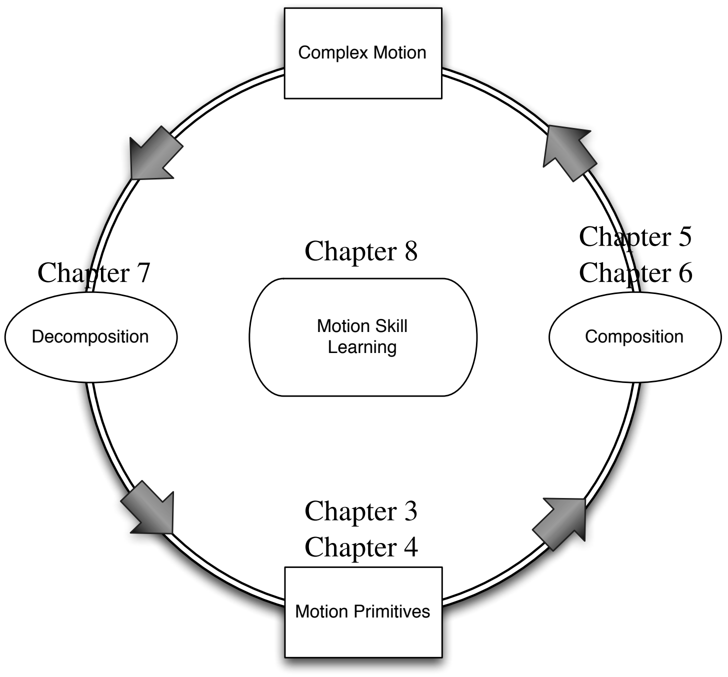


Fig. 9.1: Motion skill learning is a complex process discussed in this thesis. The endeavor starts in Chapter 3,4, where movement primitives representations are in focus of research. How to compose these movement primitives to create complex motions are introduced in Chapter 5,6. To find the considered movement primitives again in the complex movement is described in Chapter 7. In the process cycle of the composition and decomposition can be exploited for learning motion skills and is evaluated in Chapter 8.

via point tasks and is extensively evaluated and compared to trajectory predictions of the minimum jerk model. Additionally, the impact of a global representation versus a local representation for the learning process is evaluated.

In Chapter 7, a perception method is introduced which uses a priory known movement primitive library to decompose complex trajectory. This chapter contributes a heuristic method for (i) noise reduction, (ii) feature extraction depending on the primitive library. One of the appealing feature of this method is that the movement primitives processed as stereotypical shapes, which allows to use any movement generation method to generate the actual movement primitive.

Finally, a skill learning framework is introduced in Chapter 8 which uses all learning concepts and methodologies previously described. An important step towards learning new movement primitives and how to produce complex motion in a autonomous learning schema. It demonstrates that the system learns specified movement primitives, which allows to generate smooth and efficient task dependent complex motions and can also be used in other tasks, if necessary. In contrast to standard imitation learning strategies, a simple copying of the demonstrated motion is avoided. On the contrary the movement capabilities of the robot agent are used to represent the task trajectory. The limitation to only 2D trajectories is mainly due to the decomposition algorithm. However, each specific module in this learning circle can be substituted with other state of the art methods. This thesis introduced different concepts which allows to setup a learning circle that is able to efficiently learn new primitives according to the task.

CHAPTER 10

APPENDIX

Related references by the author

Contributions to conferences

[Reinhart et al., 2012] In this paper a bi-manual skill learning architecture is proposed. In this architecture movement primitives are represented as vector fields learned by the extreme learning machine without stability constraints. All authors contributed to the research design. This work contributes to the objective to the new learning methodology for the extreme learning machines with stability constraints described in Chapter 3.

[Lemme et al., 2012] This poster submission, presents first conceptual ideas for co-articulation of straight motions with one artificial neural network. All authors contributed to the research design. This work contributes to Chapter 6. I presented the poster at NCM in Venice, Italy.

[Lemme et al., 2013] This paper proposes a learning methodology to model dynamical systems with neural networks from sparse trajectory data. The paper is joint work of the European project ”AMARSi” and the leading-edge cluster “IT’s OWL”. This paper received the best student paper award at ESANN 2013 in Bruges, Belgium. All authors contributed to the research design. The concepts and results of this paper contribute to Chapter 3.

[Neumann et al., 2013a] This paper proposes a learning approach to extend the flexibility in approximating vector fields for stable dynamical systems. All authors contributed to the research design. I was involved in designing the robotic experiment and integration of the learning concepts for movement primitives. This paper related to Chapter 3.

[Soltoggio et al., 2012] In this paper the first conceptual ideas of the used decomposition algorithm are presented. All authors contributed to the research design. This paper contributes to Chapter 7.

[Khansari et al., 2013] The benchmark framework was first introduced and discussed in this workshop, and first preliminary results were presented by the first benchmark participants. M.S. Khansari-Zadeh, Y. Meirovitch, and myself were manly involved in developing the latest version of the benchmark software framework. I contributed in organizing the half-day workshop. I conducted the workshop and presented the conceptual ideas of this work in Atlanta, USA. This work contributed to Chapter 4.

[Lemme et al., 2014b] In this work the semi-supervised learning cycle to bootstrap new primitives form complex trajectories is introduced. All authors contributed to the research design. The concepts and results of this paper build the basis for Chapter 8.

Journal publications

[Soltoggio and Lemme, 2013] We extended the work introduced in [Soltoggio et al., 2012] and added more detailed experimental evaluations and an extended discussion. Both authors contributed equally to the research design. This paper contributes to Chapter 7.

[Lemme et al., 2014a] This paper extend the research done in [Lemme et al., 2013]. The results acquired in [Lemme et al., 2013] are now extensively discussed and additional experiments with the robotic platform iCub were conducted. All authors contributed to the research design. This paper is published in the special issue of ESANN 2013 conference. The concepts and results of this paper build the basis for Chapter 3.

[Lemme et al., 2015] This paper describes the concepts of the benchmark framework introduced and discussed in [Khansari et al., 2013]. All authors contributed to the research design. The concepts of this paper build the basis for Chapter 4.

REFERENCES

W. Abend, E. Bizzi, and P. Morasso. Human arm trajectory formation. *Brain : a journal of neurology*, 105(2):331–348, 1982.

R. Amit and M. Matari. Learning movement sequences from demonstration. In *Proc. Int. Conf. on Development and Learning*, pages 203–208, 2002.

B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

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.

C. Atkeson and J. M. Hollerbach. Kinematic features of unrestrained vertical arm movements. *The Journal of Neuroscience*, 5(9):2318–2330, 1985.

C. Atkeson and J. McIntyre. Robot trajectory learning through practice. In *Proc. Int. Conf. on Robotics and Automation*, volume 3, pages 1737–1742, 1986.

C. Atkeson and S. Schaal. Robot learning from demonstration. In *Proc. Int. Conf. on Machine Learning*, pages 12–20, 1997.

J. Babic, J. Hale, and E. Oztop. Human sensorimotor learning for humanoid robot skill synthesis. *Adaptive Behavior*, 19(4):250–263, 2011.

D. H. Bailey and J. T. Barton. *The NAS kernel benchmark program*. National Aeronautics and Space Administration, Ames Research Center, 1985.

M. S. Bazaraaa, H. Sherali, and C. Shetty. *Nonlinear programming: Theory and Algorithms*. Ed. John Wiley & Sons, 2006.

D. Bennequin, R. Fuchs, A. Berthoz, and T. Flash. Movement timing and invariance arise from several geometries. *PLoS computational biology*, 5(7):e1000426, 2009.

N. Bernstein. *The Coordination and Regulation of Movements*. Oxford: Pergamon Press, 1967.

N. Berthier. Learning to reach: A mathematical model. *Developmental psychology*, 32(5):811–823, 1996.

G. Biggs and B. MacDonald. A survey of robot programming systems. In *Australasian conference on robotics and automation*, pages 1–10, 2003.

A. Billard and M. J. Matarić. Learning human arm movements by imitation: Evaluation of a biologically inspired connectionist architecture. *Robotics and Autonomous Systems*, 37(2):145–160, 2001.

A. Billard, S. Calinon, R. Dillmann, and S. Schaal. *Robot Programming by Demonstration*, chapter 59, pages 1371–1394. Springer, 2008.

E. Bizzi, M.C. Tresch, P. Saltiel, and A. d'Avella. New perspectives on spinal motor systems. *Nature Reviews Neuroscience*, 1(2):101–108, 2000.

E. Bizzi, V.C.K. Cheung, A. d'Avella, P. Saltiel, and M. Tresch. Combining modules for movement. *Brain Research Reviews*, 57(1):125–133, 2008.

M. D. K. Breteler, J. M. Hondzinski, and M. Flanders. Drawing sequences of segments in 3D: Kinetic influences on arm configuration. *Journal of Neurophysiology*, 89(6):3253–3263, 2003.

D. Bullock and S. Grossberg. The VITE model: A neural command circuit for generating arm and articulator trajectories. *Dynamic patterns in complex systems*, pages 305–326, 1988.

D. Bullock and S. Grossberg. VITE and FLETE: Neural modules for trajectory formation and postural control. *Volitional action*, pages 253–298, 1989.

S. Calinon, F. Guenter, and A. Billard. On Learning, Representing, and Generalizing a Task in a Humanoid Robot. *Transactions on Systems, Man, and Cybernetics*, 37(2):286–98, 2007.

S. Calinon, T. Alizadeh, and D. G. Caldwell. On improving the extrapolation capability of task-parameterized movement models. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 610–616, 2013.

T. Chaminade, D. W. Franklin, E. Oztop, and G. Cheng. Motor interference between humans and humanoid robots: Effect of biological and artificial motion. In *Proc. Int. Conf. on Development and Learning*, pages 96–101, 2005.

S. Chiaverini. Singularity-Robust Task Priority Redundancy Resolution for Real-time Kinematic Control of Robot Manipulators. *Transactions on Robotics and Automation*, 13(3):398–410, 1997.

M. A. Conditt, F. Gandolfo, and F. A. Mussa-Ivaldi. The motor system does not learn the dynamics of the arm by rote memorization of past experience. *Journal of Neurophysiology*, 78(1):554–560, 1997.

B. Corteville, E. Aertbelien, H. Bruyninckx, J. De Schutter, and H. Van Brussel. Human-inspired robot assistant for fast point-to-point movements. In *Proc. Int. Conf. on Robotics and Automation*, pages 3639–3644, 2007.

A. d'Avella and E. Bizzi. Shared and specific muscle synergies in natural motor behaviors. *The National Academy of Sciences of the United States of America*, 102(8):3076–3081, 2005.

A. d'Avella and F. Lacquaniti. Control of reaching movements by muscle synergy combinations. *Frontiers in Computational Neuroscience*, 7(42), 2013.

A. d'Avella, P. Saltiel, and E. Bizzi. Combinations of muscle synergies in the construction of a natural motor behavior. *Nature neuroscience*, 6(3):300–308, 2003a.

A. d'Avella, P. Saltiel, and E. Bizzi. Combinaitons of muscle synergies in the construction of a natural motor behavior. *Nature Neuroscience*, 6:300–306, 2003b.

S. Degallier and A. Ijspeert. Modeling discrete and rhythmic movements through motor primitives: a review. *Biological Cybernetics*, 103(4):319–338, 2010.

Y. Demiris and B. Khadhouri. Hierarchical attentive multiple models for execution and recognition of actions. *Robotics and Autonomous Systems*, 54(5):361 – 369, 2006.

Y. Demiris and G. Simmons. Perceiving the unusual: Temporal properties of hierarchical motor representations for action perception. *Neural Networks*, 19(3):272–284, 2006.

E. D. Dolan and J. J. More. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

S. Edelman. Representation is representation of similarities. *Brain and Behavioural Sciences*, 21(4): 449–467, 1998.

S. Edelman and T. Flash. A model of handwriting. *Biologicial Cybernetics*, 57:25–36, 1987.

C. Elbrechter, M. Götting, and R. Haschke. Image Component Library (ICL), 2013. URL `http://www.iclcv.org`.

D. Endres, Y. Meirovitch, T. Flash, and M. Giese. Segmenting sign language into motor primitives with bayesian binning. *Frontiers in computational neuroscience*, 7, 2013.

P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381–391, 1954.

T. Flash. Organizing principles underlying the formation of hand trajectories. *Doctoral dissertation, Massachusetts Institute of Technology, Cambridge, MA*, 1983.

T. Flash. The control of hand equilibrium trajectories in multi-joint arm movements. *Biological cybernetics*, 57(4-5):257–274, 1987.

T. Flash and B. Hochner. Motor primitives in vertebrates and invertebrates. *Current Opinion in Neurobiology*, 15(6):660 – 666, 2005. Motor sytems / Neurobiology of behaviour.

T. Flash and N. Hogan. The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model. *Neuroscience*, 5(7):1688–1703, 1985.

T. Flash, E. Henis, R. Inzelberg, and A. D. Korczyn. Timing and sequencing of human arm trajectories: Normal and abnormal motor behaviour. *Human movement science*, 11(1):83–100, 1992.

T. Flash, Y. Meirovitch, and A. Barliya. Models of human movement: trajectory planning and inverse kinematics studies. *Robotics and Autonomous Systems*, 61(4):330–339, 2013.

C. A. Fowler and E. Saltzman. Coordination and coarticulation in speech production. *Language and speech*, 36(2-3):171–195, 1993.

M. Fumagalli, S. Ivaldi, M. Randazzo, and F. Nori, 2011. URL `http://eris.liralab.it/wiki/Force_Control`.

J. Fuster. Upper processing stages of the perception-action cycle. *Trends in Cognitive Sciences*, 8(4): 143 – 145, 2004.

K. Gaj, E. Homsirikamol, and M. Rogawski. Fair and comprehensive methodology for comparing hardware performance of fourteen round two sha-3 candidates using fpgas. In *Cryptographic Hardware and Embedded Systems*, volume 6225, pages 264–278. Springer, 2010.

F. Gandolfo, F. A. Mussa-Ivaldi, and E. Bizzi. Motor learning by field approximation. *Proceedings of the National Academy of Sciences*, 93(9):3843–3846, 1996.

M. Gharib, F. Pereira, D. Dabiri, and D. Modarress. Quantitative flow visualization. *Annals of the New York Academy of Sciences*, 972(1):1–9, 2002.

M. Gienger, H. Janssen, and C. Goerick. Task-oriented whole body motion for humanoid robots. In *Proc. Int. Conf. on Humanoid Robots*, pages 238–244, 2005.

M. Gienger, M. Toussaint, N. Jetchev, A. Bendig, and C. Goerick. Optimization of fluent approach and grasp motions. In *Proc. Int. Conf. on Humanoid Robots*, pages 111–117, 2008.

M. Giese, A. Mukovskiy, A. Park, L. Omlor, and J. Slotine. Real-time synthesis of body movements based on learned primitives. In *Statistical and Geometrical Approaches to Visual Motion Analysis*, volume 5604 of *Lecture Notes in Computer Science*, pages 107–127. Springer Berlin Heidelberg, 2009.

S. F. Giszter and W. J. Kargo. Modeling of dynamic controls in the frog wiping reflex: force-field level controls. *Neurocomputing*, 38(0):1239–1247, 2001.

J.V. Gómez, D. Alvarez, S. Garrido, and L. Moreno. Kinesthetic teaching via fast marching square. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 1305–1310, 2012.

K. Grave and S. Behnke. Incremental action recognition and generalizing motion generation based on goal-directed features. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 751–757, 2012.

C. M. Harris and D. M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394(6695): 780–784, 1998.

C. Hart and S. Giszter. A Neural Basis for Motor Primitives in the Spinal Cord. *Neuroscience*, 30(4): 1322–1366, 2010.

J. Heinzmann and A. Zelinsky. Quantitative safety guarantees for physical human-robot interaction. *The International Journal of Robotics Research*, 22(7-8):479–504, 2003.

S. Hellbach, J. P. Eggert, E. Koerner, and M. Gross. Basis Decomposition of Motion Trajectories using Spatio-Temporal NMF. In *Proc. Int. Conf. on Artificial Neural Networks*, volume 5769, pages 804–814, 2009.

N. Hogan. An organizing principle for a class of voluntary movements. *Neuroscience*, 4(11):2745, 1984.

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme Learning Machine: A new learning scheme of feedforward neural networks. In *International Joint Conference on Neural Networks*, pages 985–990, 2004.

Z. Huang and C. Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1–3): 489–501, 2006.

A. J. Ijspeert. Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653, 2008.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. Int. Conf. on Robotics and Automation*, volume 2, pages 1398–1403, 2002.

A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Proc. Advances in Neural Information Processing Systems 15*, pages 1523–1530. MIT Press, 2003.

A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

T. Inamura, Y. Nakamura, H. Ezaki, and I. Toshima. Imitation and primitive symbol acquisition of humanoids by the integrated mimesis loop. In *Proc. Int. Conf. on Robotics and Automation*, volume 4, pages 4208–4213, 2001.

T. Inamura, I. Toshima, and Y. Nakamura. Acquiring motion elements for bidirectional computation of motion recognition and generation. In *Experimental Robotics VIII*, volume 5, pages 372–381. Springer, 2003.

M. Ito and J. Tani. Generalization in learning multiple temporal patterns using RNNPB. In *Proc. Int. Conf. on Neural Information Processing*, pages 592–598, 2004.

O.C. Jenkins and M.J. Mataric. Deriving action and behavior primitives from human motion data. In *IROS*, volume 3, pages 2551–2556, 2002.

A. Karniel and F. A. Mussa-Ivaldi. Does the motor control system use multiple models and context switching to cope with a variable environment? *Experimental Brain Research*, 143(4):520–524, 2002.

M. S. Khansari, A. Lemme, Y. Meirovitch, B. Schrauwen, M. A. Giese, A. J. Ijspeert, A. Billard, and J. J. Steil. Workshop on benchmarking of state-of-the-art algorithms in generating human-like robot reaching motions. In *Proc. Int. Conf. on Humanoid Robots*, 2013.

S. M. Khansari-Zadeh. `http://www.amarsi-project.eu/open-source`, 2012a.

S. M. Khansari-Zadeh. *A Dynamical System-based Approach to Modeling Stable Robot Control Policies via Imitation Learning*. PhD thesis, EPFL, 2012b.

S. M. Khansari-Zadeh. `http://lasa.epfl.ch/people/member.php?SCIPER=183746/`, 2013.

S. M. Khansari-Zadeh and A. Billard. BM: An iterative algorithm to learn stable non-linear dynamical systems with Gaussian mixture models. In *Proc. Int. Conf. on Robotics and Automation*, pages 2381–2388, 2010a.

S. M. Khansari-Zadeh and A. Billard. Imitation learning of globally stable non-linear point-to-point robot motions using nonlinear programming. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 2676–2683, 2010b.

S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with Gaussian mixture models. *Transactions on Robotics*, 27(5):943–957, 2011.

S. M. Khansari-Zadeh and A. Billard. Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions robotics and autonomous systems. *Robotics and Autonomous Systems*, 62(6):752 – 765, 2014.

J. Kohlmorgen and S. Lemm. A Dynamic HMM for On-line Segmentation of Sequential Data. In *Proc. NIPS*, volume 14, pages 793–800, 2001.

G. Konidaris, S. Kuindersma, A. Barto, and R. Grupen. Constructing skill trees for reinforcement learning agents from demonstration trajectories. In *Proc. Advances in Neural Information Processing Systems.*, volume 23, pages 1162–1170, 2010.

G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012.

D. Kulic, L. Dongheui, C. Ott, and Y. Nakamura. Incremental learning of full body motion primitives for humanoid robots. In *Proc. Int. Conf. on Humanoid Robots*, pages 326–332, 2008a.

D. Kulic, W. Takano, and Y. Nakamura. Combining automated on-line segmentation and incremental clustering for whole body motions. In *Proc. Int. Conf. on Robotics and Automation*, pages 2591–2598, 2008b.

D. Kulic, W. Takano, and Y. Nakamura. Online segmentation and clustering from continuous observation of whole body motions. *Transactions on Robotics*, 25(5):1158–1166, 2009.

D. Kulic, C. Ott, L. Dongheui, J. Ishikawa, and Y. Nakamura. Incremental learning of full body motion primitives and their sequencing through human motion observation. *The International Journal of Robotics Research*, 31(3):330–345, 2011.

T. Kulvicious, K. Ning, M Tamosiunaite, and F. Woergoetter. Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *Transactions on Robotics*, 28:145–157, 2012.

Y. Kuroe and H. Kawakami. Vector field approximation by model inclusive learning of neural networks. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 717–726, 2007.

Y. Kuroe and H. Kawakami. Estimation method of motion fields from images by model inclusive learning of neural networks. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 673–683, 2009.

F. Lacquaniti, C. Terzuolo, and P. Viviani. The law relating kinematic and figural aspects of drawing movements. *Acta Psychologica*, 54:115–130, 1983.

K. Lashley. The problem of serial order in behavior. *Cerebral mechanisms in behavior. New York: Wiley*, pages 112–135, 1951.

Mark L Latash, John P Scholz, and Gregor Schöner. Toward a new theory of motor synergies. *Motor control*, 11(3), 2007.

Dongheui Lee and Christian Ott. Incremental motion primitive learning by physical coaching using impedance control. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 4133–4140, 2010.

A. Lemme, Y. Meirovitch, T. Flash, and J. J. Steil. Co-articulation of straight lines with an artificial neural network. In *Neural Control of Movement (NCM)*, 2012. Poster.

A. Lemme, K. Neumann, F. R. Reinhart, and J. J. Steil. Neurally imprinted stable vector fields. In *Proc. ESANN*, pages 327–332, 2013. Best paper award.

A. Lemme, Neumann K., F. R. Reinhart, and J. J. Steil. Neural learning of vector fields for encoding stable dynamical systems. *Neurocomputing*, 141(0):3–14, 2014a.

A. Lemme, R. F. Reinhart, and J. J. Steil. Self-supervised bootstrapping of a movement primitive library from complex trajectories. In *Proc. Int. Conf. on Humanoid Robots*, page 726–732, 2014b.

A. Lemme, Y. Meirovitch, M. S. Khansari, Tamar T., A. Billard, and J. J. Steil. Multi-criteria benchmarking of movement generating dynamical systems for learning-from-demonstrations. *Paladyn. Journal of Behavioral Robotics*, 6(1):30–41, 2015.

N. Liang, G. Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *Transactions on Neural Networks*, 17(6):1411–1423, 2006.

A. Liégeois. Automatic Supervisory Control of Configuration and Behavior of Multibody Mechanisms. *Transactions on Systems, Man and Cybernetics*, 7(12):861–871, 1977.

T. Luksch, M. Gienger, M. Mühlig, and T. Yoshiike. A dynamical systems approach to adaptive sequencing of movement primitives. In *German Conference on Robotics*, pages 1–6, 2012a.

T. Luksch, M. Gienger, M. Mühlig, and T. Yoshiike. Adaptive movement sequences and predictive decisions based on hierarchical dynamical systems. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 2082–2088, 2012b.

T. Luksch, M. Gienger, M. Mühlig, and T. Yoshiike. A dynamical systems approach to adaptive sequencing of movement primitives. In *Proc. German Conf. on Robotics*, pages 1–6, 2012c.

B. Luokkala. Can a machine become self-aware? In *Exploring Science Through Science Fiction*, pages 81–103. Springer, 2014.

R. Mann, A. Jepson, and T. El-Maraghi. Trajectory segmentation using dynamic programming. In *Proc. Int. Conf. on Pattern Recognition*, volume 1, pages 331 – 334, 2002.

F. Meier, E. Theodorou, F. Stulp, and S. Schaal. Movement segmentation using a primitive library. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 3407–3412, 2011.

F. Meier, E. Theodorou, and S. Schaal. Movement segmentation and recognition for imitation learning. In *Proc. Int. Conf. on Artificial Intelligence and Statistics*, pages 761–769, 2012.

A. Merz, K. Peterson, and K. Riedl. *Zebra 1, Buchstabenheft in Grundschrift*. Klett, 2012.

V. Mohan, P. Morasso, J. Zenzeri, G. Metta, S. Chakravarthy, and G. Sandini. Teaching a humanoid robot to draw 'Shapes'. *Autonomous Robots*, 31(1):21–53, 2011.

Vishwanathan Mohan, Giorgio Metta, Jacopo Zenzeri, and Pietro Morasso. Teaching humanoids to imitate shapes of movements. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 234–244. Springer, 2010.

M. Moran. The da vinci robot. *Journal of endourology*, 20(12):986–990, 2006.

P. Morasso. Spatial control of arm movements. *Experimental Brain Research*, 42(2):223–227, 1981.

F. Moro, N. Tsagarakis, and D. G. Caldwell. A human-like walking for the COmpliant huMANoid COMAN based on CoM trajectory reconstruction from kinematic Motion Primitives. In *Proc. Int. Conf. on Humanoid Robots*, pages 364–370, 2011.

F. L. Moro, Nikos G. Tsagarakis, and D. G. Caldwell. On the kinematic motion primitives (kmps) - theory and application. *Frontiers in Neurorobotics*, 6(10), 2012.

M. Mühlig, M. Gienger, J. J. Steil, and C. Goerick. Automatic selection of task spaces for imitation learning. In *IEEE/RSJ Proc. Int. Conf. on Intelligent Robots and Systems*, pages 4996–5002, 2009.

M. Mühlig, M. Gienger, and J. J. Steil. Interactive imitation learning of object movement skills. *Autonomous Robots*, 32(2):97–114, 2012.

A. Mukovskiy, Slotine J.J.E., and M. Giese. Dynamically stable control of articulated crowds. *Journal of Computational Science*, 4(4):304–310, 2013.

S. Murata, J. Namikawa, H. Arie, S. Sugano, and J. Tani. Learning to reproduce fluctuating time series by inferring their time-dependent stochastic properties: Application in robot learning via tutoring. *Transactions on Autonomous Mental Development*, PP(99):1–1, 2013.

F.A. Mussa-Ivaladi and Bizzi E. Motor learning through the combination of primitives. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 355(1404):1755–1769, 2000.

F. A. Mussa-Ivaldi. From basis functions to basis fields: vector field approximation from sparse data. *Biological Cybernetics*, 67(6):479–489, 1992.

F. A. Mussa-Ivaldi and S. F. Giszter. Vector field approximation: a computational paradigm for motor control and learning. *Biological Cybernetics*, 67(6):491–500, 1992.

F. A. Mussa-Ivaldi, S. Giszter, and E. Bizzi. Linear combinations of primitives in vertebrate motor control. *National Academy of Sciences*, 91(16):7534–7538, 1994.

C. Nehaniv and K. Dautenhahn. Of hummingbirds and helicopters: an algebraic. *Interdisciplinary Approaches to Robot Learning*, 24:136, 2000.

K. Neumann, A. Lemme, and J. J. Steil. Neural learning of stable dynamical systems based on data-driven lyapunov candidates. In *Proc. Int. Conf. of Intelligent Robots and Systems (IROS)*, pages 1216–1222, 2013a.

K. Neumann, M. Rolf, and J. J. Steil. Reliable integration of continuous constraints into extreme learning machines. *Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 21(supp02):35–50, 2013b.

K. Nigam and R. Ghani. Analyzing the effectiveness and applicability of co-training. In *Proc. Int. Conf. on Information and Knowledge Management*, CIKM '00, pages 86–93. ACM, 2000.

E. Oztop, D. Franklin, T. Chaminade, and G. Cheng. Human–humanoid interaction: Is a humanoid robot perceived as a human? *International Journal of Humanoid Robotics*, 2(04):537–559, 2005.

R. Paine and J. Tani. Motor primitive and sequence self-organization33 in a hierarchical recurrent neural network. *Neural Networks*, 17:1291–1309, 2004.

A. Paraschos, G. Neumann, and J. Peters. A probabilistic approach to robot trajectory generation. In *Proc. Int. Conf. on Humanoid Robots*, pages 477–483, 2013.

D. Park, H. Hoffmann, P. Pastor, and S. Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *Proc. Int. Conf. on Humanoid Robots*, pages 91–98, 2008.

A. Parmiggiani, M. Randazzo, L. Natale, G. Metta, and G. Sandini. Joint torque sensing for the upper-body of the iCub humanoid robot. In *Proc. Int. Conf. on Humanoid Robots*, pages 15–20, 2009.

P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal. Learning and generalization of motor skills by learning from demonstration. In *Proc. Int. Conf. on Robotics and Automation*, pages 763–768, 2009.

P. Pastor, M. Kalakrishnan, F. Meier, F. Stulp, J. Buchli, E. Theodorou, and S. Schaal. From dynamic movement primitives to associative skill memories. *Robotics and Autonomous Systems*, 61(4):351 – 361, 2013.

U. Pattacini, F. Nori, L. Natale, G. Metta, and G. Sandini. An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots. In *IROS*, pages 1668–1674. IEEE, 2010.

M. Paulus. How and why do infants imitate? an ideomotor approach to social and imitative learning in infancy (and beyond). *Psychonomic Bulletin & Review*, pages 1–18, 2014.

J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008.

A. Pistillo, S. Calinon, and D. G. Caldwell. Bilateral physical interaction with a robot manipulator through a weighted combination of flow fields. In *Proc. Int. Conf. on Intelligent Robots and Systems*, pages 3047–3052, 2011.

F. Pollick, U. Maoz, A. Handzel, P. Giblin, G. Sapiro, and T. Flash. Three-dimensional arm movements at constant equi-affine speed. *Cortex*, 45(3):325–339, 2009.

F. Polyakov, E. Stark, R. Drori, M. Abeles, and T. Flash. Parabolic movement primitives and cortical states: merging optimality with geometric invariance. *Biological Cybernetics*, 100(2):159–184, 2009.

F. R. Reinhart and J. J. Steil. Neural learning and dynamical selection of redundant solutions for inverse kinematic control. In *Proc. Int. Conf. on Humanoid Robots*, pages 564–569, 2011.

F. R. Reinhart, A. Lemme, and J. J. Steil. Representation and generalization of bi-manual skills from kinesthetic teaching. In *Proc. Int. Conf. on Humanoid Robots*, pages 560–567, 2012.

R. F. Reinhart. Reservoir computing with output feedback. *KI-Künstliche Intelligenz*, 26(4):415–416, 2012.

B. Rhodes, D. Bullock, W. Verwey, B. Averbeck, and M. Page. Learning and production of movement sequences: Behavioral, neurophysiological, and modeling perspectives. *Human movement science*, 23(5):699–746, 2004.

M. Riley, A. Ude, K. Wade, and C. Atkeson. Enabling real-time full-body imitation: A natural way of transferring human movement to humanoids. In *Proc. Int. Conf. on Robotics and Automation*, volume 2, pages 2368–2374, 2003.

B. Rohrer and N. Hogan. Avoiding spurious submovement decompositions: a globally optimal algorithm. *Biological cybernetics*, 89(3):190–199, 2003.

M. Rolf, J. J. Steil, and M. Gienger. Efficient exploration and learning of whole body kinematics. In *Proc. Int. Conf. on Development and Learning*, pages 1–7, 2009.

M. Rolf, J. J. Steil, and M. Gienger. Online goal babbling for rapid bootstrapping of inverse models in high dimensions. In *Proc. Int. Conf. of Development and Learning and on Epigenetic Robotics*, 2011.

C. Rosenberg, M. Hebert, and H. Schneiderman. Semi-supervised self-training of object detection models. In *WACV/MOTIONS*, pages 29–36, 2005.

T. Sanger. Failure of motor learning for large initial errors. *Neural Computation*, 16(9):1873–1886, 2004.

S. Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6): 233–242, 1999.

S. Schaal. Dynamic movement primitives -a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*, pages 261–280. Springer, 2006.

S. Schaal and C. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998.

S. Schaal and N. Schweighofer. Computational motor control in humans and robots. *Current Opinion in Neurobiology*, 15(6):675 – 682, 2005.

S. Schaal and D. Sternad. Programmable pattern generators. In *Proc. Int. Conf. on Computational Intelligence in Neuroscience*, pages 48–51. Citeseer, 1998.

S. Schaal, A. Ijspeert, and A. Billard. Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547, 2003a.

S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Control, planning, learning, and imitation with dynamic movement primitives. In *Proc. Int. Conf. on Intelligent Robots and Systems*, 2003b.

S. Schaal, D. Sternad, R. Osu, and M. Kawato. Rhythmic arm movement is not discrete. *Nature neuroscience*, 7(10):1136–1143, 2004.

S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In *Robotics Research*, volume 15, pages 561–572. Springer, 2005.

T. Schack. The cognitive architecture of complex movement. *International Journal of Sport and Exercise Psychology*, 2(4):403–438, 2004.

R. Shadmehr and F. A. Mussa-Ivaldi. Adaptive representation of dynamics during learning of a motor task. *The Journal of Neuroscience*, 14(5):3208–3224, 1994.

A. Soltoggio and A. Lemme. Movement primitives as a robotic tool to interpret trajectories through learning-by-doing. *International Journal of Automation and Computing*, 10(5):375–386, 2013.

A. Soltoggio, A. Lemme, and J. J. Steil. Using movement primitives in interpreting and decomposing complex trajectories in learning-by-doing. In *Proc. Int. Conf. on Robotics and Biomimetics*, pages 1427 – 1433, 2012.

R. Sosnik, B. Hauptmann, A. Karni, and T. Flash. When practice leads to co-articulation: the evolution of geometrically defined movement primitives. *Experimental Brain Research*, 156:422–438, 2004.

R. Sosnik, T. Flash, B. Hauptmann, and A. Karni. The acquisition and implementation of the smoothness maximization motion strategy is dependent on spatial accuracy demands. *Experimental brain research*, 176(2):311–331, 2007.

T.H. Speeter. Primitive based control of the utah/mit dextrous hand. In *Proc. Int. Conf. on Robotics and Automation*, volume 1, pages 866–877, 1991.

D. Sternad and S. Schaal. Segmentation of endpoint trajectories does not imply segmented control. *Experimental Brain Research*, 124(1):118–136, 1999.

F. Stulp and S. Schaal. Hierarchical reinforcement learning with movement primitives. In *Proc. Int. Conf. on Humanoid Robots*, pages 231–238, 2011.

J. Tani, M. Ito, and Y. Sugita. Self-organization33 of distributedly represented multiple behavior schemata in a mirror system: reviews of robot experiments using RNNPB. *Neural Networks*, 17 (8-9):1273–1289, 2004.

E. Todorov. Optimality principles in sensorimotor control. *Nature Neuroscience*, 7:907–915, 2004.

E. Todorov and M. I. Jordan. Smoothness maximization along a predefined path accurately predicts the speed profiles of complex arm movements. *Neurophysiology*, 80:696–714, 1998.

M. Tresch, V. Cheung, and A. d'Avella. Matrix Factorization Algorithms for the Identification of Muscle Synergies: Evaluation on Simulated and Experimental Data Sets. *Journal of Neurophysiology*, 95(4): 2199–2212, 2006.

N.G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A.J. Ijspeert, M.C. Carrozza, et al. iCub: the design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics*, 21(10):1151–1175, 2007.

A. Ude, C. Atkeson, and M. Riley. Programming full-body movements for humanoid robots by observation. *Robotics and Autonomous Systems*, 47(2-3):93–108, 2004.

A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives. *Transactions on Robotics*, 26(5):800–815, 2010.

A. Verri and T. Poggio. Motion field and optical flow: qualitative properties. *Transactions on Pattern Analysis and Machine Intelligence*, 11(5):490–498, 1989.

P. Viviani and M. Cenzato. Segmentation and coupling in complex movements. *Journal Experimental Psychology Humam Perception and Performence*, 11(6):828–845, 1985.

P. Viviani and C. deSperati. The relationsheep between curvature and velocity in two dimensional smooth persuit eye movement. *The Journal of Neuroscience*, 17:3932–3945, 1997.

P. Viviani and R. Schneider. A developmental study of the relationship between geometry and kinematics in drawing movements. *Journal of Experimental Psychology: Human Perception and Performance*, 17(1):198–218, 1991.

Y. Wada and M. Kawato. A via-point time optimization algorithm for complex sequential trajectory formation. *Neural networks*, 17(3):353–364, 2004.

Y. Wada, Y. Koike, E. Vatikiotis-Bateson, and M. Kawato. A computational theory for movement pattern recognition based on optimal movement pattern generation. *Biological Cybernetics*, 73(1):15–25, 1995.

Yasuhiro Wada and Mitsuo Kawato. A theory for cursive handwriting based on the minimization principle. *Biological Cybernetics*, 73(1):3–13, 1995. ISSN 0340-1200. doi: 10.1007/BF00199051.

D. Wolpert and Z. Ghahramani. Computational principles of movement neuroscience. *nature neuroscience*, 3:1212–1217, 2000.

D. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7):1317–1329, 1998.

D. Wolpert, Z. Ghahramani, and M. Jordan. Are arm trajectories planned in kinematic or dynamic coordinates? An adaptation study. *Experimental brain research*, 103(3):460–470, 1995.

Y. Yamashita and J. Tani. Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. *PLoS Comput Biol*, 4(11):e1000220, 2008.