Franz Steinmetz

Programming by Demonstration for in-contact tasks using Dynamic Movement Primitives

School of Electrical Engineering

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, 19.8.2014

Thesis supervisors:

Prof. Ville Kyrki
Prof. Thomas Gustafsson

Thesis instructor:

Ph.D. Alberto Montebelli

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE MASTER'S THESIS

Author: Franz Steinmetz

Title: Programming by Demonstration for in-contact tasks using Dynamic Movement Primitives

Date: 19.8.2014 Language: English Number of pages: 9+74

Department of Electrical Engineering and Automation

Professorship: Automation Technology Code: AS-84

Supervisors: Prof. Ville Kyrki, Prof. Thomas Gustafsson

Instructor: Ph.D. Alberto Montebelli

Despite the rapid growth in the number of robots in the world, the number of service robots is still very low. The major reasons for this include the robots' lack of world knowledge, sensitivity, safety and flexibility. This thesis experimentally addresses the last three of these issues (sensitivity, safety and flexibility) with reference to advanced, industrial level robotic arms provided with integrated torque sensors at each joint.

The aims of this work are twofold. The first one, at a more technical level, is the implementation of a real-time software infrastructure, based on Orocos and ROS, for a general, robust, flexible and modular robot control framework with a relatively high level of abstraction. The second aim is to utilize this software framework for Programming by Demonstration with a class of algorithms known as Dynamic Movement Primitives. Using kinesthetic teaching with one or multiple demonstrations, the robot performs simple sequential in-contact tasks (e. g. writing on a notepad a previously demonstrated sequence of characters). The system is not only able to imitate and generalize from demonstrated trajectories, but also from their associated force profiles during the execution of in-contact tasks. The framework is further extended to successfully recover from perturbations during the execution and to cope with dynamic environments.

Keywords: Programming by Demonstration, Dynamic Movement Primitives, In-contact tasks

Preface

The thesis was carried out in the Intelligent Robotics group at the Aalto University, in which I met only great people. In this group, I especially want to thank my professor Ville Kyrki and my instructor Alberto Montebelli. Ville gave me the opportunity to work on this great project, supported me in choosing my own directions for the thesis topic and was always available for questions. Alberto helped me not only to critically think about several aspects of the thesis and to greatly improve the language and style of the thesis. With our common private activities he also distracted me from the sometimes stressful work and introduced me together with his girlfriend Kathri to delicious cuisines during fantastic evenings.

I received great financial support from the European Space Agency (ESA) Directorate of Human Spaceflight and Operation, for which I want to gratefully thank them.

As part of the SpaceMaster program, I got to know many fantastic, interesting, funny, international colleagues, with whom I spend an amazing time. Especially those who also chose Helsinki for their second year (Christina, Michael, Nemanja, Oliver & Tesfamichael) became good friends of mine. They always had a friendly ear, we had great discussions and a fantastic time together. Thank you, guys!

On top of that, I received a lot of support from home. I want to thank my parents for their always supportive words. Also all my friends earn gratitude, to whom I always could talk to.

Finally, I would be nothing without my girlfriend Judith. Thank you for supporting me all the time and helping me to overcome our abiding long-distance relationship. I love you!

Espoo, 19.8.2014 Franz Steinmetz

Contents

Abstract ii

Preface iii

Contents iv

Symbols and abbreviations vi

1 Introduction 1

2 Programming by Demonstration 3
- 2.1 Teaching methods . . . 3
- 2.2 Symbolic learning . . . 3
- 2.3 Trajectory learning . . . 4
  - 2.3.1 Traditional encoding . . . 5
  - 2.3.2 Statistical models and dynamic systems . . . 5
- 2.4 Comparison of learning approaches . . . 13
  - 2.4.1 Qualitative evaluation . . . 14
  - 2.4.2 Quantitative evaluation . . . 18
- 2.5 Discussion . . . 20
- 2.6 Alternatives to Programming by Demonstration . . . 21

3 Dynamic Movement Primitives 23
- 3.1 Basics of Dynamic Movement Primitives . . . 23
  - 3.1.1 Execution phase . . . 23
  - 3.1.2 Learning phase . . . 28
  - 3.1.3 Multiple dimensions . . . 30
- 3.2 Dynamic Movement Primitives for in-contact tasks . . . 30
  - 3.2.1 Controller . . . 30
  - 3.2.2 Handling of perturbations . . . 31
  - 3.2.3 Handling of desired forces . . . 33

4 Infrastructure 35
- 4.1 Overview . . . 35
- 4.2 Hardware . . . 36
- 4.3 Fast Research Interface . . . 38
- 4.4 KUKA Robot Language . . . 39
- 4.5 Middleware: ROS and Orocos . . . 39
- 4.6 Software architecture . . . 40
  - 4.6.1 Basic framework . . . 41
  - 4.6.2 Recording . . . 42
  - 4.6.3 Imitation . . . 43
  - 4.6.4 Implementation details . . . 45
- 4.7 Discussion . . . 46

5 Experiments **48**
5.1 Imitation of a circular motion 49
5.1.1 Experiment 49
5.1.2 Results 50
5.2 Multiple demonstrations 51
5.2.1 Experiment 51
5.2.2 Results 52
5.3 Perturbation handling 53
5.3.1 Experiment 53
5.3.2 Results 53
5.4 Writing on a notepad 54
5.4.1 Experiment 54
5.4.2 Results 55
5.5 Writing on a lowered notepad 58
5.5.1 Experiment 58
5.5.2 Results 59
5.6 Discussion 62

6 Conclusion **64**
6.1 Future work 66

References **68**

Appendices **73**

A Class diagram **73**

Symbols and abbreviations

Symbols

$A_{\max}$	maximum amplitude of a demonstration
c_i	center of the Gaussian or von Mises function i
C_c	coupling term of the canonical system
C_c, stop	temporal coupling term needed to stop the canonical system
C_t	coupling term of the transformation system
$D(d_\text{c})$	Cartesian damping term (dependent on the standardized damping value)
$D(d_\text{j})$	joint damping term (dependent on the standardized damping value)
Δe^j	absolute error
$\Delta e^j_{\max}$	maximum allowed absolute error
$e_{\text{max, rel}}$	maximum of the relative errors
$e'_{\text{max, rel}}$	maximum of the relative errors (limited to one)
e^j_{rel}	absolute relative error
$\mathbf{f}$	stacked force vector for all demonstrations
$\mathbf{f}^j$	force vector for demonstration j
$f(x)$	function approximator
f_{contact}	threshold force value to be in contact (in Z direction)
f_{des}	desired Cartesian forces and torques
$f_{\text{des, min}}$	minimum desired force to activate force controller
$f_{\text{dyn}}(q, \dot{q}, \ddot{q})$	Dynamic arm forces (gravity, Coriolis, etc.)
f_{DMP}	desired force calculate from DMP (in Z direction)
f_{msr}	measured force (in Z direction)
g	goal of a trajectory
h	measure of concentration for the von Mises function
i	index for basis function and weighting factor
j	index for demonstration
J^T	transposed Jacobian
k_c	Cartesian stiffness parameter
k_j	joint stiffness parameter
M_j	number of trajectory points for demonstration j before trimming
M'_j	number of trajectory points for demonstration j after trimming
N_D	number of demonstrations
N_w	number of basis functions
q_{des}	desired joint positions
q_{msr}	measured joint positions
$\mathbf{s}$	stacked scaling vector for all demonstrations
$\mathbf{s}^j$	scaling vector for demonstration j
t	time
t_{learn}	Learning time
v_{approach}	Desired approach velocity
v_{cur}	Current approach velocity
w_i	weighting factor i

$\dot{x}$	temporal derivative of the phase variable
x	phase variable
x_0	initial value of the phase variable
x_{des}	desired Cartesian position
x_{msr}	measured Cartesian position
y	position
$\dot{y}$	velocity
$\ddot{y}$	acceleration
α_x	decay factor of the discrete canoncial system
α_z	stiffness of the transformation spring-damper-system
β_z	damping of the transformation spring-damper-system
Ψ_i	basis function i
$\mathbf{\Psi}_i$	compound diagonal basis function matrix for weighting factor i
$\mathbf{\Psi}_i^j$	diagonal basis function matrix for demonstration j and weighting factor i
σ	standard deviation of the Gaussian function
τ	time scaling factor (duration of execution)
τ_{cmd}	joint torques generated by the controller
τ_{des}	desired joint torques
$\xi(x)$	scaling term

List of acronyms

BIC Bayesian Information Criterion

CAD computer-aided design

DMP Dynamic Movement Primitives

DoF degrees of freedom

DTW Dynamic Time Warping

DP Dynamic programming

EM Expectation-Maximization

FRI Fast Research Interface

F/T force/torque

GMM Gaussian mixture model

GMR Gaussian mixture regression

GPR Gaussian process regression

HMM Hidden Markov Model

HRI human-robot interaction

KCP KUKA Control Panel

KRC KUKA Robot Controller

KRL KUKA Robot Language

LGP Local Gaussian process

LOOCV Leave One Out Cross Validation

LSOGP Localized Sparse Online Gaussian Process

LWL Locally weighted learning

LWPR Locally Weighted Projection Regression

LWR light-weight robot

LWR Locally weighted regression

MNS mirror neuron system

MSE mean square error

NURBS Non-Uniform Rational B-Splines

OOP Object-oriented programming

Orocos Open Robot Control Software

PbD Programming by Demonstration

PCBC piecewise cubic Bézier curves

PFM Potential Field Method

PID controller proportional-intergral-derivative controller

RL reinforcement learning

RMS root mean square

ROS Robot Operating System

SEDS Stable Estimator of Dynamical Systems

UDP User Datagram Protocol

1 Introduction

Robots can hardly be found in any household. There may be lawn mowing or vacuum cleaning robots. However, they are very limited and specific both in their functionality and interaction with humans. Two reasons for that are the demanding challenges of *human-robot interaction* (HRI) and the complexity of household tasks. For these complex tasks, robots typically miss some important attributes: world knowledge, sensitivity (capability to sense environmental stimuli), safety and flexibility. This thesis work tries to improve the situation for the last three of these attributes.

Programming by Demonstration (PbD), also known as *imitation learning*, is a broad field that tries to cope with problems at a similar level of complexity as the mentioned household tasks [1]. A human teacher is directly integrated in the robot learning process. The teacher demonstrates a skill to the robot, which receives the information via various sensors and sensor modalities. An algorithm tries to extract an appropriate representation of the demonstration, which must contain the key features of the experienced skill. Crucially, such a representation must be capable of generalization, i. e. performing adequately in the case of perturbations or changes in the environment. The aim of this thesis is to teach (by demonstration) robots human-level skills with the necessary sensitivity for *in-contact tasks* [2].

In-contact tasks Usually in-contact tasks (in the field of robotics) are understood as tasks, which require a contact between the robot and the environment. The object of contact is typically static, but can also be dynamic, animated or not (for example a human patient during a surgery). In-contact tasks therefore require force exertion between the two objects being in contact. These forces must be controlled to both fulfill the tasks and not to damage anything or harm anyone. In this thesis, the definition of in-contact tasks is extended to undesired contacts. During the execution of a task, non desired contact is possible in a dynamic environment (for example when a person steps into the work area). These contacts also need appropriate handling, for the sake of the proper task execution and safety. An example for an in-contact task using PbD is given in Figure 1. It shows the setup of the experiment that will be described in Section 5.4 and 5.5, in which the instructor directly demonstrates the robot to write on a notepad.

In-contact tasks have been studied for over 30 years [3]. Especially the dynamic interaction forces of in-contact tasks have been investigated in detail (e. g. [4]). In-contact tasks were have been in the context of grasping [5] or for strategies to reach a certain contact configuration [6].

More relevant to the contents of this thesis are the studies about impedance controllers. A breakthrough for in-contact tasks may have been [7] – "Stiffness isn't everything". Before this study, the target of most robot actuator designs was the maximization of the stiffness. However, [7] demonstrates the advantages of a compliant design for in-contact tasks. This is supported by [8], in which the importance of impedance control for in-contact tasks for humans is highlighted. In [8], it is shown how humans adapt the stiffness of their muscolo-skeletal-system depending on the requirements of the task. The stiffness is dynamically changed right before starting

the motion for a task. These insights are for example used in [9]. There, the dynamic stiffness of a muscle is imitated using a pneumatic actuator. The stiffness is controlled by the air pressure.

Despite all of these studies, literature about learning impedance and contact forces is very limited. In [10], the optimal impedance is learned for a number of practical experiments, such as door-opening. However, the impedance is not learned from demonstrations, but using *reinforcement learning* (RL) (see Section 2.6). The paper that is more closely related to this thesis is [11]. Both force and stiffness are learned from kinesthetic teaching (see Section 2.1). The success of their methods are shown in an ironing and an door-opening experiment.

Objectives This work aims for three main objectives. First, by comparing current PbD approaches, the one most appropriate for the context of this thesis is searched. Second, a software framework should be established, in which this PbD approach is implemented in a flexible manner. Third, the algorithms on the chosen PbD approach should be expanded to make them applicable for in-contact tasks, which means that both trajectories and force profiles should be imitated. The approach should be applied to an existing robotic arm equipped with torque sensors in each joint. Thus the system is static (not mobile), but works in a dynamic environment with users. The safety of the users is very important. The system should proof its capabilities in an actual robotics experimental scenario, in which the robot learns to write letters.

Structure The thesis is organized as follows. In Chapter 2, more information about PbD and its alternatives is offered. In this chapter, also the different learning approaches are described and evaluated in detail. The following Chapter 3 describes the chosen *Dynamic Movement Primitives* (DMP) approach in detail. First, the existing basics about this framework are given (Section 3.1). Then, these principles are extended by novel algorithms for in-contact tasks (Section 3.2). The infrastructure used for the project is presented in Section 4. There, also the newly developed software architecture is introduced (Chapter 4). Chapter 5 experimentally evaluates the developed DMP framework. Finally, an overall conclusion is given in Section 6.

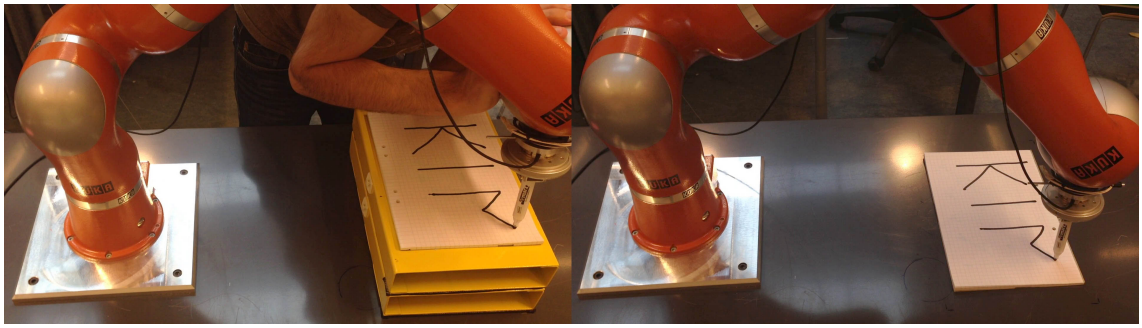

Figure 1: Setup of the writing experiments. In the left image, the trajectory is recorded using kinesthetic teaching (see Section 2.1). The right image shows the (autonomous) execution phase with the notepad directly positioned on the table.

2 Programming by Demonstration

This chapter explores the details of PbD. First, the different teaching methods are introduced. This is followed by an extensive review of PbD learning approaches. Finally, alternatives to PbD are given.

2.1 Teaching methods

Imitation learning itself can technically be done using three different methods, each with different features [1, 12]. The first is *teleoperation*, in which the user has some kind of remote controller or haptic device and can control the robot from a distant position. The second method is *kinesthetic teaching*; in this case, the user directly moves the robot by grabbing it. This approach can be more intuitive for the operator, who might feel more natural in the execution of the demonstration and get a better impression about the robots limitations. The third is *observational learning*, that uses cameras or other positioning/motion capture systems; the movement of a user is recorded and transferred to the robot. In this case, the different embodiments must be considered, an issue that is also known as the correspondence problem [1].

Kinesthetic teaching is gaining importance due to improved hardware of state-of-the-art robots [2]. In order to manually move a robot, its motors must be either backdrivable or actively controlled [11]. While the first method has been existing for a longer time, it is also more limited. Gear transmission cause friction, which as well as the inertia of bigger robots require higher forces, thus restraining the teacher in the execution of natural movements. Actively controlled robots are often much more sophisticated and include *force/torque* (F/T)-sensors in each joint or for all Cartesian axes. In combination with advanced controllers, these systems allow a very easy and natural steering. Most recent developments of *light-weight robots* (LWRs), such as the KUKA LWR [13], support gravity compensation and dynamically changeable compliance.

Another advantage of kinesthetic teaching is the possible recording of force profiles. For this, an additional F/T-sensor is mounted at the tip or wrist of the manipulator. However, also haptic devices have been used in teleoperation for force feedback [11]. By integrating these measures, not only positional/trajectory data are collected from the instructor, but also force and torque profiles when being in-contact with objects. This information constitutes a crucial component of the dynamics of in-contact skills [2].

2.2 Symbolic learning

Learning algorithms in research usually either take place on the symbolic or trajectory level. While the latter is explained in detail in the next section (2.3), the former is only outlined briefly here. Symbolic learning is much about reasoning. Questions being answered are more of the form “what” and “why” to execute a certain action, instead of “how” to do it.

In [14], the typical building blocks of PbD at symbolic level are presented (see Figure 2). A demonstrated task must first be segmented, i. e. discrete actions have to be identified. A typical example is setting up a dinner table, which (arguably) consists of several pick-and-place actions. After that, the actions are modeled as states, taking into account several slightly varying demonstrations. An important aspect of this kind of research is the following step of task generalization, which tries to derive a temporal and hierarchical order of the states. Referring to the previous example, the saucer must be placed before the cup, but these are independent from the cutlery. Having planned the task, the robot may execute the task on its own, though the scope of this execution phase is still very limited and not handled on the symbolic level. Another big challenge is the perception side, which is both needed for the learning and the planning/execution part.

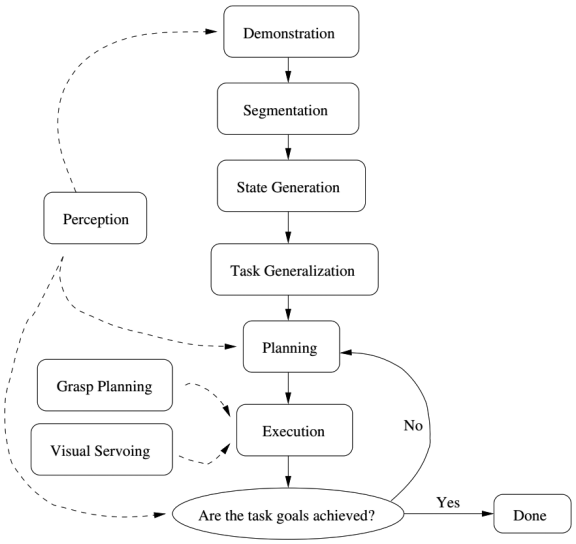

Figure 2: Building blocks of a PbD approach on the symbolic level. (Source: [14])

A review over the issues of perception and recognition is given in [15]. Mostly video cameras are used in combination with motion capture and tracking algorithms. Recognition is divided in four approaches in [15]: Scene-based (only the position of objects is followed without knowing their identity), full-body-based (humans and their positions are identified), body-parts-based (human limb actions are considered) and action-primitives-based (actions are split into their primitives and interpreted).

On the one hand, this approach has the advantage of being abstract and high-level, making it possible to teach whole tasks. On the other hand, this approach has a drawback: Many things have to be predefined (for example possible actions) and the robot has to be provided with a broad knowledge about the world [1].

2.3 Trajectory learning

Learning on the trajectory level is currently more relevant for practical implementations compared to symbolic learning. Trajectory level gives rise to the question,

how to perform a given task. The used methods have evolved from strict trajectory encoding to representations using statistical models or dynamical systems.

2.3.1 Traditional encoding

Traditionally trajectories are encoded using *splines* or *Bézier curves* [16, 17]. Thus the encoding is done explicitly, the generated trajectory is defined from a fixed starting point to a fixed endpoint.

An example for traditional encoding is given in [16]. Triangulation with two cameras is used to record trajectories from human instructors. Different models are derived to calculate the poses. The poses include uncertainty by using Gaussian distributions, whose variances are considered when generating vector spline trajectories. The trajectories have the constraints of being smooth and as close to the presented trajectory as possible, according to a given metric.

In [17], two more feature were introduced. First, a *significant point extraction algorithm* was developed, which chooses less points than common corner detection algorithms. Second, their generated *piecewise cubic Bézier curves* (PCBC) are online adaptable to obstacles. Figure 3 shows schematically the steps of this approach. The success of their approach is demonstrated with a 2 D mobile robot. This robot can be steered using a touchpad. During reproduction in a cluttered environment, obstacles are avoided automatically.

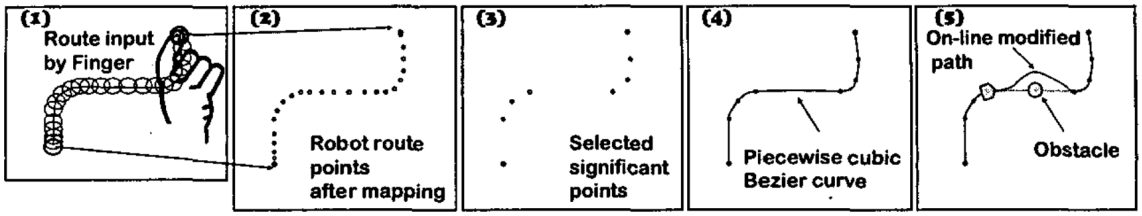

Figure 3: After extracting significant points from a given route, a PCBC is extracted and can be modified online. (Source: [17])

An approach to learn from several demonstrations is shown in [18]. The shown examples are first clustered according to their distance to each other. Next *Hidden Markov Models* (HMMs) are used to derive *Non-Uniform Rational B-Splines* (NURBS). An example is shown in Figure 4.

2.3.2 Statistical models and dynamic systems

Besides this traditional encoding approach, a number of various recent methods exist. They can hardly be categorized as each has their own features and disadvantages (see Section 2.4 for a comparison). These different approaches make use of statistical models or dynamical systems (or both) and can often be described as regression methods.

Regression is an important part of statistics and a number of different regression methods have been adapted to fit the needs of PbD. Informally speaking, regression

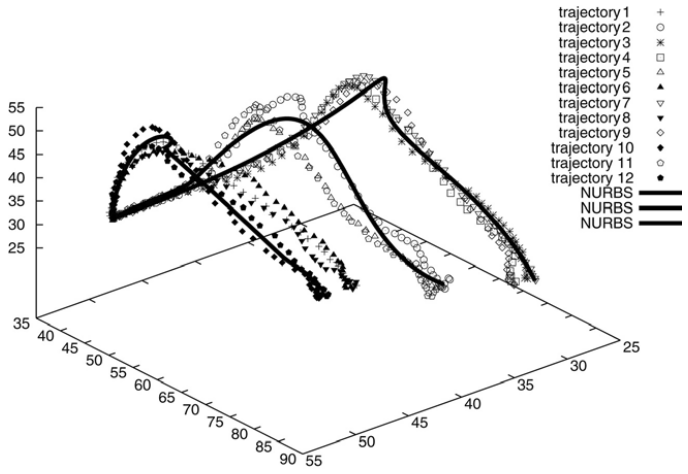

Figure 4: Similar demonstrated trajectories are clustered and converted to curves (bold). (Source: [18])

is the technique to generate a sought output (such as a position or velocity) for a given input (such as the time or the current position) based on a finite set of data. For PbD, regression is used for a plausible trajectory generation based on several demonstrations.

Locally weighted learning

Locally weighted learning (LWL) is a family of local regression methods, which are usually linear. "Local" here refers to the fact, that for the searched output, only data points, which are close (according to some metric) to the given input, are considered. For regression, only these points are weighted depending on their distance [19], see Figure 5. The first LWL approaches, named *Locally weighted regression* (LWR), were fully memory-based, i. e. all data points were kept and used for regression calculations. A LWL implementation is characterized for example by its distance functions, weighting functions, smoothing parameters and more; a good overview is given in [20]. Figure 6 exemplifies outlier detection/removal, which can be another feature.

LWL methods have been expanded to be incremental (thus not memory-based) and computationally efficient for high dimensional spaces [21]. Following the approach named *Locally Weighted Projection Regression* (LWPR), LWL is thus able to learn online, even in the case of big datasets. Four approaches are experimentally compared in [21], and the LWPR method introduced there proved fast and accurate.

Gaussian process regression

The theory behind *Gaussian process regression* (GPR) is comprehensively described in [22]. A Gaussian process generalizes Gaussian distributions by defining the distribution over functions, that is a mean function and a covariance function.

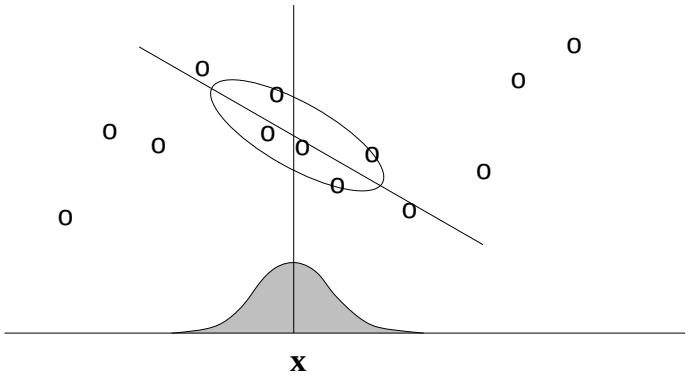

Figure 5: LWL uses only the points in the region of interest to perform local regression. (Source: [19])

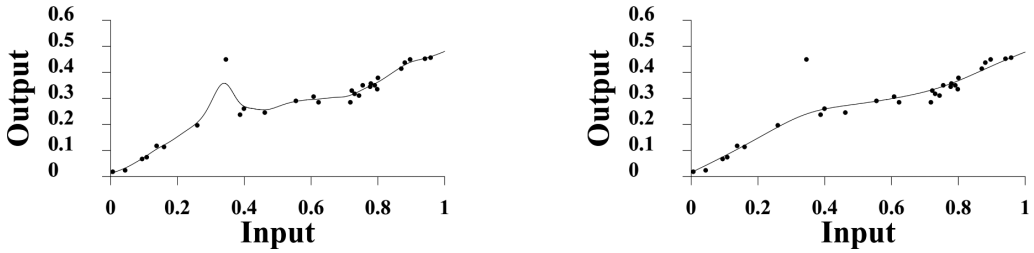

Figure 6: Two examples of LWR on a 1 D data set: The right one uses outlier removal, the left one is not. (Source: [20])

The mean and covariance functions are determined based on the input datapoints. An example is shown in Figure 7.

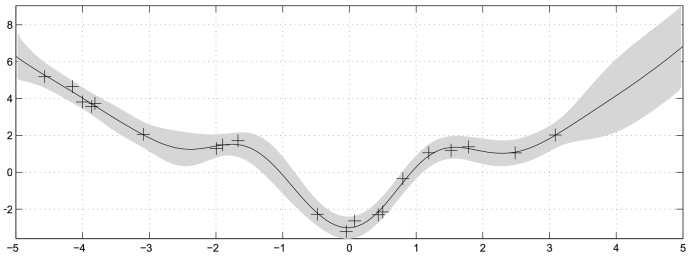

Figure 7: GPR visualized: From a finite set of data points (crosses), the continuous mean (solid line) and variance (gray area shows 90 % confidence interval) are derived. (Source: [22])

The largest drawback of plain GPR is the high computational complexity, as it lacks online learning capabilities. Thus, GPR becomes very time consuming for an increasing number of datapoints. This is due to the fact, that GPR has a computational complexity of $O(n^3)$ with n being the number of data points [22]. A reduction of the computational complexity was therefore achieved in [23] by introducing some approximation inspired from the local nature of LWL. Their resulting *Local Gaussian process* (LGP) models proves more computational efficient ($O(n^3/M)$ with M

being the number of subspaces) while maintaining the same levels of accuracy. The approach assigns new data points to local models. These are extracted by a distance metric using a Gaussian kernel.

A recent extension using sparse approximation was presented in [24]. Sparse approximations are another technique to speed up computations. Instead of looking at local sub-data as in the previous approach, only data points that are considered most salient according to a certain metric are kept, thus reducing the number of data points. In [24], *Leave One Out Cross Validation* (LOOCV) is used to determine the representational value of new data. In combination with local regression, this constitutes *Localized Sparse Online Gaussian Process* (LSOGP).

Gaussian mixture regression

Gaussian mixture regression (GMR) consists of several (partly optional) steps, which are shown in Figure 8 and are subsequently explained. In order to obtain values using GMR, first a probabilistic model is needed. Usually, either the HMM or the *Gaussian mixture model* (GMM) are used, which share many properties. For a comparison of the two (based on temporal alignment), see [25]. Both GMM and HMM are described in the present section. These models usually operate in the task space, that typically consits of the (relative) Cartesian position and velocity. The raw measurements, once projected in the task space, form the data set which consists of datapoints.

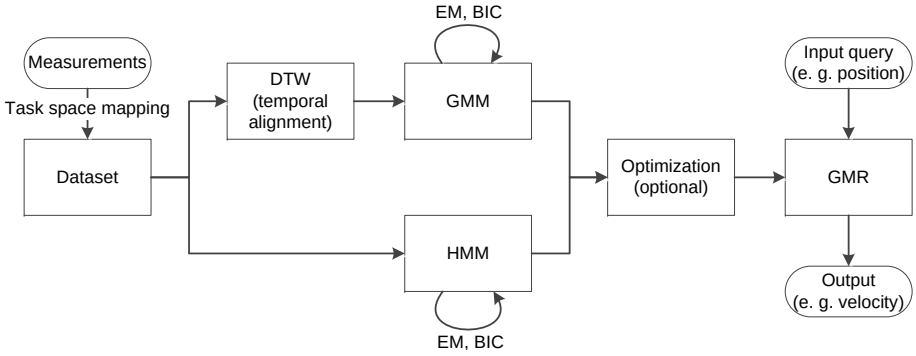

Figure 8: Building blocks of GMR: After a model is generated (GMM or HMM) and optimized, GMR is used to retrieve commands.

As the GMM also models time, *temporal normalization* among the available demonstrations is needed. This is due to the possible variation in the execution time of different demonstrations. The use of raw data would lead to an inaccurate model with high variances, see Figure 9 (a). One elegant way to achieve temporal alignment is the *Dynamic Time Warping* (DTW) algorithm, which is described in [26]. The basic idea is to align all data samples of all trajectories to achieve a pairwise minimal distance to each other, according to some distance metric. *Dynamic programming* (DP) can be used to generate the optimal solution, which is shown in Figure 9 (b).

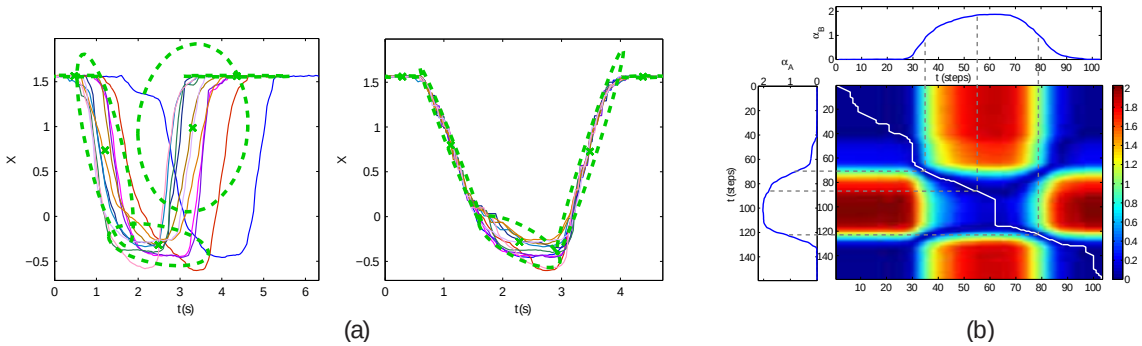

Figure 9: (a) Trajectories and resulting Gaussian model (green ovals) before (left) and after DTW (right)
(b): The white lines shows the result of DP to find the shortest path between two demonstrations; the colored center visualizes the distance matrix between the two. (Source: [26])

A GMM consists of K Gaussians of dimensionality D, which model the given data set [25]. D is the number of dimensions of the datapoints plus one, as also time is explicitly considered. Each Gaussian is described by a mean vector and a covariance matrix. The K Gaussians can be imagined as several multidimensional "bubbles" forming a sequence along the trajectory (see Figure 10). Each Gaussian is also characterized by a *prior*, which represents an initial likelihood without taking into consideration any input data. Theory and algorithms for the computation of prior, mean and covariance in GMM have been described in [19]. An advantage of Gaussian mixtures is that input and output do not need to be specified. Any subset of dimensions can be used as input, the remaining dimensions represent the output.

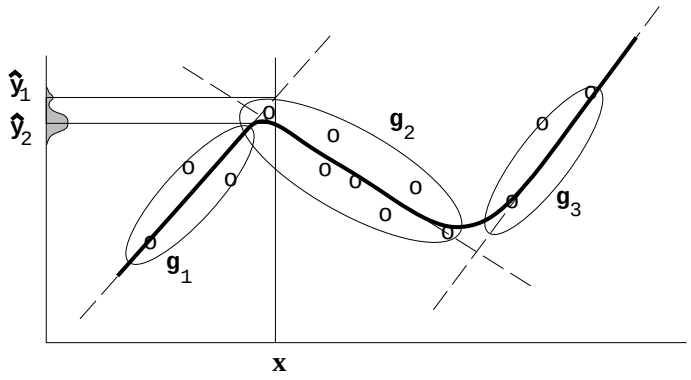

Figure 10: Gaussians model the data density. For a given input, the Gaussians are mixed according to their expectation. (Source: [19])

Similarly, HMM also consists of a mixture of Gaussians, called *states*. In addition to prior, mean and covariance, HMMs also consist of a transition matrix, describing the pair-wise transition probability to switch from one state to another. In HMMs, time is not an explicit component of task space and model.

In order for the Gaussians to effectively model the data set, some kind of training

algorithm is needed. The *Expectation-Maximization* (EM) algorithm is often used [26]. The *Baum-Welch* algorithm is a possible alternative [27]. EM iteratively adapts the parameters of all Gaussians to maximize their probability of modelling all datapoints.

Both GMM and HMM require the choice of the number of Gaussians to be used. Many different approaches for this have been described in literature. However, the approach being used most often is the *Bayesian Information Criterion* (BIC) [25, 26]. First the model is optimized for a various number of Gaussians. BIC then weights the quality of the model (in terms of probability) against the number of Gaussians. The model with the lowest BIC number is used, which yields the best compromise between accuracy and complexity. In order to avoid the computation of the exact model for many different K of Gaussians, [26] approximates the models without EM. This approach significantly reduces the computational overhead, while achieving comparable results (see Figure 11).

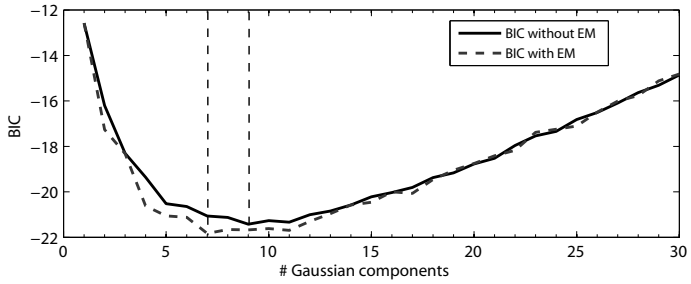

Figure 11: Calculation of the BIC value for several numbers of Gaussian, the blue line uses EM, the red line does not. (Source: [26])

Once the model is available, GMR can finally be used to obtain the desired outputs for a given input. The Gaussians that constitute the model can be interpreted as attractors, which offer a compact representation of the movement. For GMMs, the input is time [25, 26], whereas for HMMs it is the current task space configuration [27]. The output of the two models describes the task space vector and the task space target position and velocity, respectively. The corresponding covariance matrices are also provided. The output is calculated by linear combination of the conditional expectation of the input for each Gaussian component. For GMM, the weighting of each Gaussian is based only on their probability. For HMM, it is expanded by the transition probability based on the previous state.

A crucial situation can occur when the initial position is too far away from the demonstrated starting positions: the trajectory may not follow the intended one, see red line in Figure 12 (a). In [27], this problem is addressed by introducing a spring-damper-system, which forces the trajectory to converge to the desired one. The disadvantage is that this may lead to oscillations and a distorted dynamic, when the gains of the spring-damper-system are not carefully chosen. The improvement can be seen in Figure 12 (b).

The use of a similarity criterion in an additional optimization step also allows a closer reproduction of the demonstrations. This optimization step is introduced in [26] and follows the EM step. The similarity criterion is used for a gradient-based

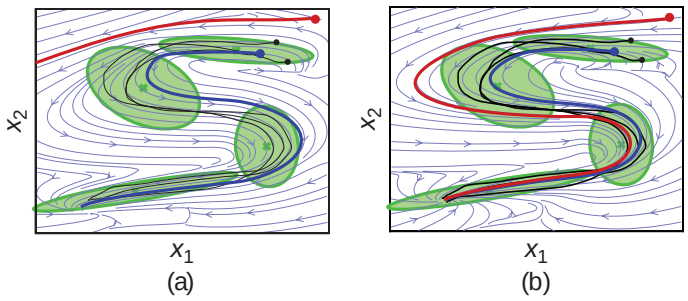

Figure 12: The figure shows the influence of a spring-damper-system to a 2 D GMR. The black lines are the demonstrations, the blue and red lines are outputs with different starting positions. (b) uses a spring-damper-system, (a) does not. (Source: [27])

optimization, as described in [28]. Further examples which can be included in the cost function are distances to obstacles or joint limit proximities.

Stable Estimator of Dynamical Systems

Stable Estimator of Dynamical Systems (SEDS) is a novel learning method producing a globally stable system [29, 30]. SEDS also relies on Gaussian mixtures for the modeling part and generates motions that closely imitate the demonstration. However, its focus lies on ensuring global asymptotic stability at the target, in contrast to the other approaches described earlier.

Global stability is considered here in the terms of motion generation in the task space. In other words, a motion generating function is defined as *globally asymptotically stable*, if motions starting from any point in the task space converge asymptotically to a given goal vector. Conversely, *local asymptotic stability* is ensured when the motion starting from a subspace of the task space converges asymptotically [29].

The model of SEDS consists again of a mixture of Gaussians. Only position and velocity is taken into account (as for HMM), but no transition matrix exists. Following a formal proof, sufficient criteria for global stability are identified and the model's parameters (prior, mean, covariance) can be accordingly constrained [29, 30]. Finally, SEDS determines the values for the constrained parameters by solving an optimization problem either using *log-likelihood* or *mean square error* (MSE). The above mentioned BIC is used to choose the optimal number of Gaussians. The parameters of SEDS (mean, covariance) are visualized in Figure 13.

Dynamic Movement Primitives

Similar to GMR, DMP is more a framework or design principle, rather than a set of defined equations and steps (see Section 3.1 for implementation details). The approach consists of three main parts, a *canonical system*, a *nonlinear function approximator* and a *transformation system* [31, 32, 33]. These basic components are briefly described in the following paragraphs and shown in Figure 14. A more

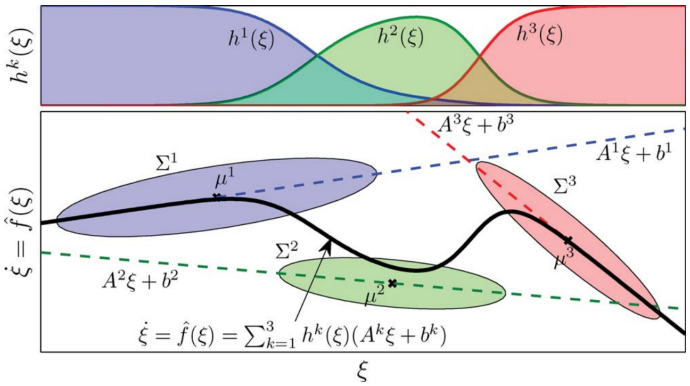

Figure 13: GMM for a 1 D position input ξ and 1 D velocity output $\dot{\xi}$. The thick solid line describes the function $\hat{f}$ determining a velocity depending on the positional input. In the upper part, the influence of the three Gaussians on $\hat{f}$ is shown, depending on ξ. (Source: [30])

thorough presentation of DMPs will follow in Chapter 3. DMP allow both discrete (point-to-point) and rhythmic (limit cycle or cyclic) motions.

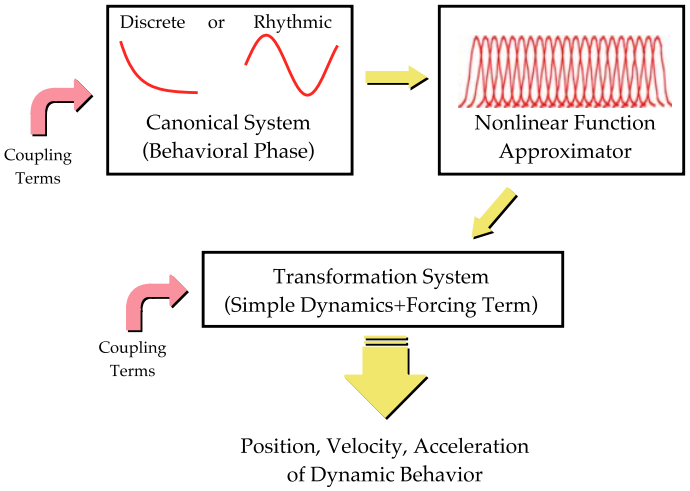

Figure 14: Building blocks of DMP (Source: [32])

The transformation system is a stable dynamical system that converts an input force and goal position into output commands such as position, velocity and acceleration. It is typically implemented as a critically damped spring-damper-system (which can be seen as a point attractor) and a forcing term. The output is then typically fed into a low-level controller that generates the motor commands. Additional coupling terms allow online modulation of the spatial trajectory.

The nonlinear function approximator is used to modify the attractor landscape by exciting the transformation system with a force term. This is done to achieve the desired trajectories rather than plain asymptotic convergence. The input of this function is the phase variable. Any function approximator can be utilized but typically a normalized linear combination of distribution functions (Gaussians, von

Mises) is used. This combination of basis functions is modulated by the phase variable and by an amplitude. The phase variable, monotonically decreasing to zero, causes the influence of the force term to vanish to guarantee stability. The amplitude serves as scaling term, allowing the trajectory to be spatially stretched or shrinked while ensuring the invariance properties of the system.

The third part of DMP is the canonical system, which is a substitute for time. In its simplest form, it is a first-order linear system generating a phase variable. This phase variable decreases exponentially from an arbitrary start value to zero. For rhythmic motions, a phase oscillator can be used, consisting of amplitude and phase signal. In both cases, a coupling term can be introduced, which allows for online modulation in the temporal space.

Usually, the trajectory has a dimension greater than one. To incorporate several *degrees of freedom* (DoF) (or dimensions), each dimension retrieves its own transformation system and forcing term, while all dimension share the same canonical system. By this, the canonical system couples the different DoF and can be seen as central clock. [33] highlights similarities with biological pattern generators.

One advantage of DMP is the number of options in learning the parameters for the function approximator to obtain the desired trajectory. Goal, amplitude and time scaling can easily be derived from the shown demonstrations. The remaining parameters, e. g. the weighting parameters for the basis functions, can be calculated by LWR [33], as shown in Section 3.1.2. Thus all features of LWR are inherited, such as its computational efficiency. More importantly, incremental learning is possible. In [32], it is shown how RL (see Section 2.6) can be used to optimize the appropriate parameters. Also other optimization techniques can be incorporated to minimize jerk and torque change or optimize according to other criteria [31].

In [11, 34], the DMP approach is adapted to deploy some remarkable features. The function approximator and the transformation system are replaced by a mixture of PD-controllers, thus all dimensions share the same controller. This allows taking into account correlation between the dimensions. In addition, the variance of the demonstrations can be used to employ dynamical stiffness, i. e. forcing the robot to be compliant in positions where the variance was high and being stiff where the demonstrations showed less variance.

2.4 Comparison of learning approaches

Several different learning approaches have been presented in the previous two sections, each with different features and drawbacks. The requirements are manifold and thus are the criteria for evaluation.

The applicability of high-level symbolic approaches is still quite limited; much research in this field is still basic research. These methods are not yet mature. Nevertheless they are promising to develop as a high-level controller for the trajectory level approaches in the future.

On the trajectory level itself, the traditional encoding with splines or similar alternatives is limited. Although sophisticated algorithms are used for the model generation, the model itself currently lacks flexibility. Only positions are encoded

(over time), but no other variable from the task space. Spline encoding may still be relevant for simple tasks in industries, such as welding. Yet in research, this approach has greatly lost its relevance.

Methods using statistical models and dynamical systems are in general much more flexible. Thus this comparison hereon focuses on these approaches. In order to compare them, a qualitative and quantitative evaluation is given in the following subsections.

2.4.1 Qualitative evaluation

The capabilities of an approach can often be assessed by considering what has been or can be accomplished with it. This provides a first impression, which is then refined in the quantitative evaluation.

General aspects

First the approaches shall be generally evaluated according to their handling of time, online learning capabilities and global stability.

Time handling Implementation and handling of time varies throughout the different methods. Often, explicit time dependency is not desirable. For LWL and GPR, time can just be an additional dimension, but is often not modeled, as it has no special meaning [21, 24]. For GMR with GMM, DTW has to be performed, and time is used explicitly as input [25]. This is problematic in case of perturbations. For GMR with HMM, time is just used for the generation of the model states [27]. After that, commands are queried by the current position. The situation is very similar for SEDS [30]. DMP follows another approach [33]. Different demonstrations have to be aligned, but after that time can be discarded. The length of an execution can be set arbitrarily. Time remains as an implicit input, but methods to cope with perturbations have been developed [33].

Online learning In order to refine learned models by new demonstrations, it is desired that the additional information is quickly incorporated without the overhead of recalculations based on all data. This is called online learning. For LWL and GPR there exist variations that can do so, namely LWPR [21] and LSOGP [24], respectively. Online learning is not supported by GMR and SEDS [27, 30]. For DMP, online learning capability depends on the function approximator, but is generally possible, as described in [33].

Global stability Global stability guarantees that the movement asymptotically converges to the goal from any position in the task space (see Section 2.3.2). This is not achieved by most methods. In [29] and [30] stability is investigated for different approaches. As can be seen in Figure 15, non-stable approaches create unintended trajectories for points outside of the demonstrated region. The result are either

spurious attractors or diverging trajectories. LWL, GPR and GMR are neither locally nor globally stable. Stability is only guaranteed by SEDS [29, 30] and DMP [33].

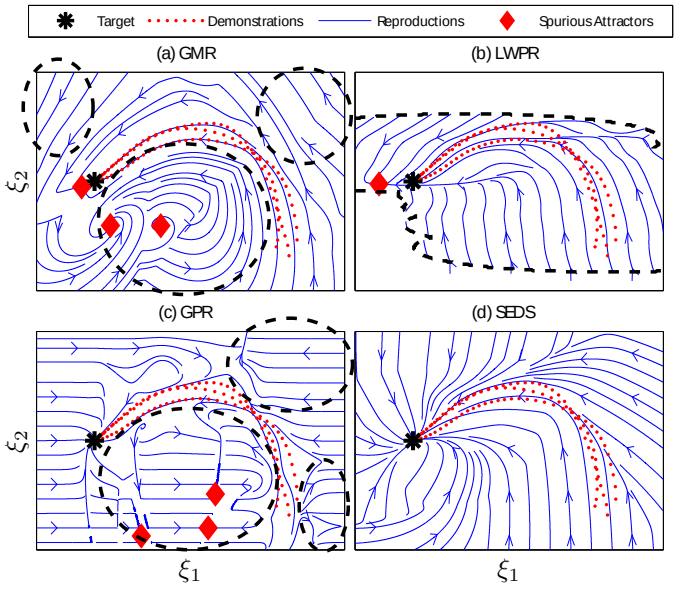

Figure 15: Depending on the stability of an approach, starting from arbitrary points may lead to diverging trajectories or spurious attractors. (Source: [29])

Specific features

In this section the most salient features of each approach are highlighted and experiments documented in robotics literature are presented.

Locally weighted learning In the field of PbD, not many examples of LWL can be found, though it is widely used in other fields. [21] describes shortly how their 7 DoF robotic arm learned to balance a pole on its fingertip from human demonstrations. Yet the main application of LWL in robotics is the learning of inverse kinematics [19, 21], also for torque control [23]. In all cases, the used LWPR method could be used for controlling in real-time ($\sim$1 ms cycle time). In general, LWR has a very broad band of applications in several research fields, such as economics, analytical chemistry and many more [20]. Additionally, LWR and its latest incarnation LWPR is used in comparison with other statistical and dynamical approaches for the reproduction of human-like motion data [27] or handwriting motions [29, 30]. For these purposes, LWPR has to be equipped with a controller, in [27] a mass-spring-damper was used.

Gaussian process regression The situation for GPR is similar to that of LWL, though these approaches were developed much more recently. To get competitive results with GPR, the improved variations LGP and LSOGP have to be used,

as they are much less computational demanding. In [23] the inverse kinematics of a 7 DoF robotic arm are quickly and accurately learned. In the same paper, it was shown that LGP can also be used for real-time torque control. LSOGP is used in [24] to demonstrate its speed and accuracy for two different inverse kinematics learning examples. Handwriting motion experiments have also been evaluated with GPR, unfortunately only with the basic variant and no local optimization [29, 30].

Gaussian mixture regression In contrast to the previous methods, GMR has been often used for the purpose of PbD. In [25] three different experiments with a human-like robot are shown. In each experiment, such as moving a chess figure, the task is demonstrated several times by the user, who manually moves the robot. The robot can generalize the task for different initial object positions. The humanoid robot ASIMO is used in [26], where the robot imitates pouring a liquid from one glass into another. A stereo vision system is used for the learning phase in the task space. Optimization is used for an accurate reproduction without self-collisions, which are a result of the correspondence problem. Even more sophisticated results were achieved with the HMM in [27]. Due to the transition matrix, it is possible to have cyclic crossing movements (see Figure 16), where the target velocity is not only dependent on the current position but also on the previous state. Self-crossing motions are not possible when only considering the position, as the trajectory contains some positions (the crossing points) more than once. Another feature of this matrix is the possibility to encode several motions in one HMM. In [27], two different table tennis strokes (top spin and drive) were demonstrated several times in arbitrary order. The two different motions were automatically seperated, as shown in Figure 17. The third experiment in [27] demonstrates how trajectories can be learned that are constrained by multiple landmarks.

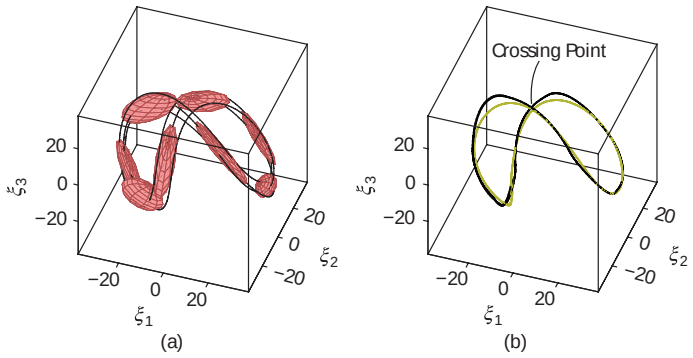

Figure 16: Cyclic motion with crossing point: (a) shows the demonstrations (black lines) and HMM states (red), (b) displays the reproduction attempts. (Source: [27])

Stable Estimator of Dynamical Systems As already mentioned, SEDS focuses on global stability. Next to the ensured convergence to the goal position, this also has some interesting "side effects" [29, 30]. First of all, demonstrated trajectories are followed very closely, as can be seen in Figure 18. Second, one SEDS model can

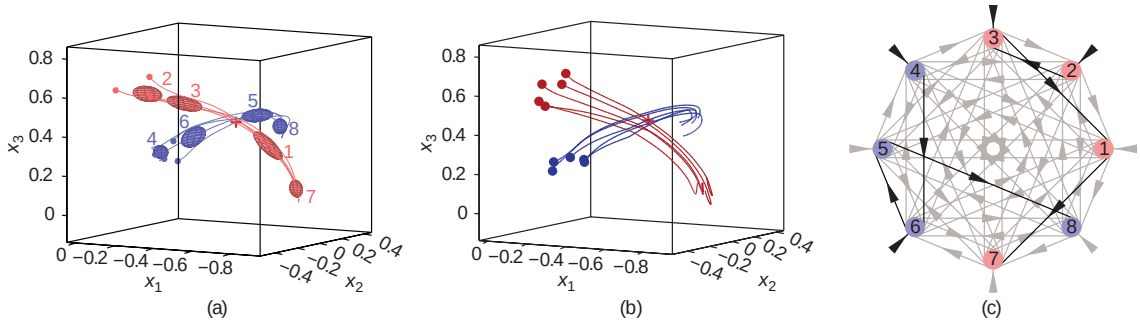

Figure 17: Learning of different movements simultaneously: (a) shows the demonstrations and the learned HMM states, (b) shows the reproductions. In (c) all possible state transitions are shown; those with a likelihood greater than 0.1 are black, as well as all likely initial states (shown with arrows). (Source: [27])

contain different movements depending on the start position, as shown in Figure 19. In addition, with SEDS, second-order dynamics can be learned to allow self-intersecting trajectories as shown for GMR. Online adaptivity in case of moving goals is also possible.

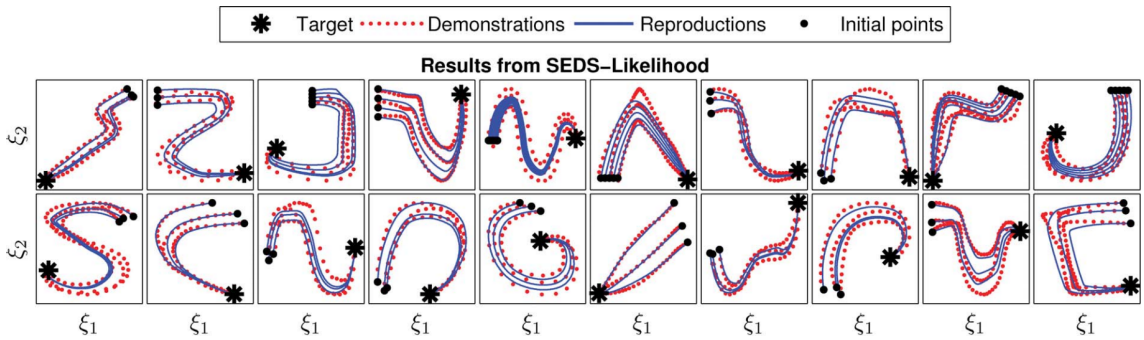

Figure 18: With SEDS, the handwriting demonstration are followed very accurately. (Source: [30])

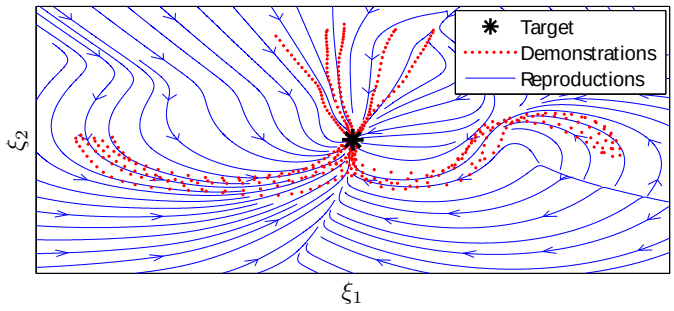

Figure 19: Different types of movements can be incorporated in one SEDS model. (Source: [29])

Dynamic Movement Primitives With DMP, similar advanced experiments can be performed. DMP is as well as the two previous methods able to follow self-crossing trajectories by extending the task-space with second-order dynamics [31]. By changing only one parameter, a learned trajectory can be amplified, sped up/slowed down or moved in space, respectively [31], as shown in Figure 20. Due to the fact that similar movements share similar weighting parameters, it is possible to perform movement segmentation and recognition [32, 33].

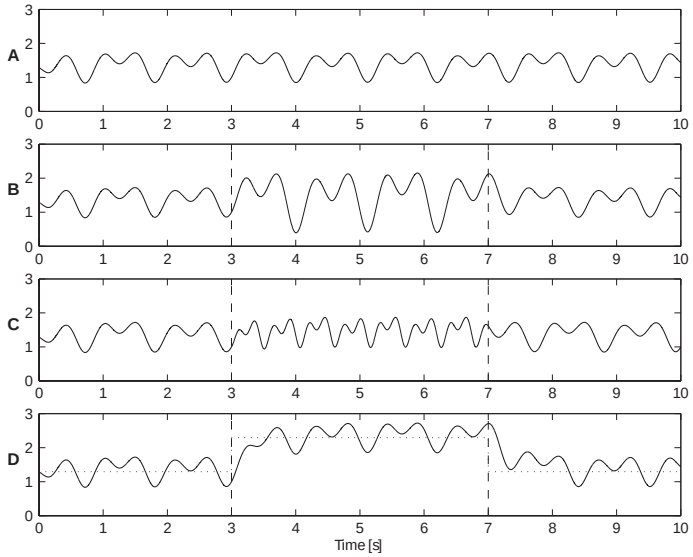

Figure 20: A learned drumming motion (A) is modified in B–C between 3 s and 7 s. In B, the amplitude is doubled, in C the time parameter is halfed. In D, the goal is increased by 1. (Source: [31])

As already mentioned, coupling terms can be introduced in the different parts of DMP to allow further extensions [32, 33]. Spatial coupling can be utilized for obstacle avoidance using force field approaches from *Potential Field Methods* (PFMs), see Figure 21. Temporal coupling can be used for rhythmical canonical systems to phase lock with an external oscillator or even to adjust to its frequency. An application is the adaption to the music beat for a drumming motion. When incorporating both spatial and temporal coupling, an elegant way to reject perturbation can be implemented. For example, when a robotic arm, following a given trajectory, is stopped for a certain time, these properties can be used to avoid high forces and speeds as the arm catches up with the expected trajectory.

With the extended DMP approach in [11, 34], it is possible to deploy what [32] describes as associative skill memories (ASM). Next to spatial information, also the force profile can be tracked in the learning process. By this, the robot is able to perform in-contact tasks such as ironing and door-opening.

2.4.2 Quantitative evaluation

The aforementioned approaches have been evaluated quantitatively in many papers. However, no publication currently covers all of these approaches up to their most

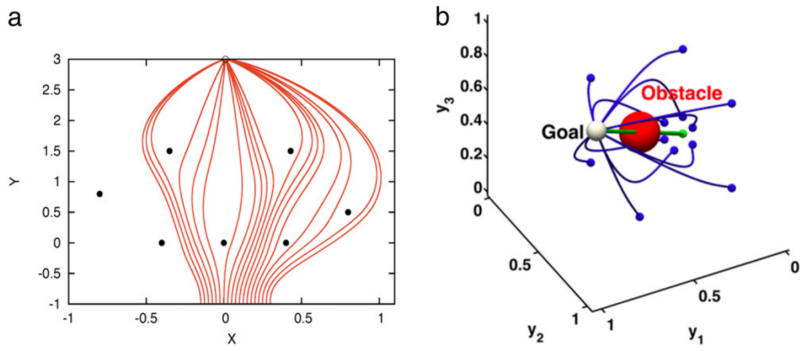

Figure 21: 2 D (a) and 3 D example (b) of DMP using PFM for obstacle avoidance. (Source: [32])

recent implementation. Therefore the results have to be taken from different sources and may thus not be fully comparable.

The most detailed comparison is done in [27], comprising GMR with HMM, GMR with GMM (there called TGMR), LWL with LWPR and DMP. Human-like motion data was artificially generated and each approach learned the movement with an increasing number of states. The comparison was then done based on the *root mean square* (RMS) error of the trajectory before and after DTW, norm of jerk, learning time and retrieval time. The results are presented in Figure 22.

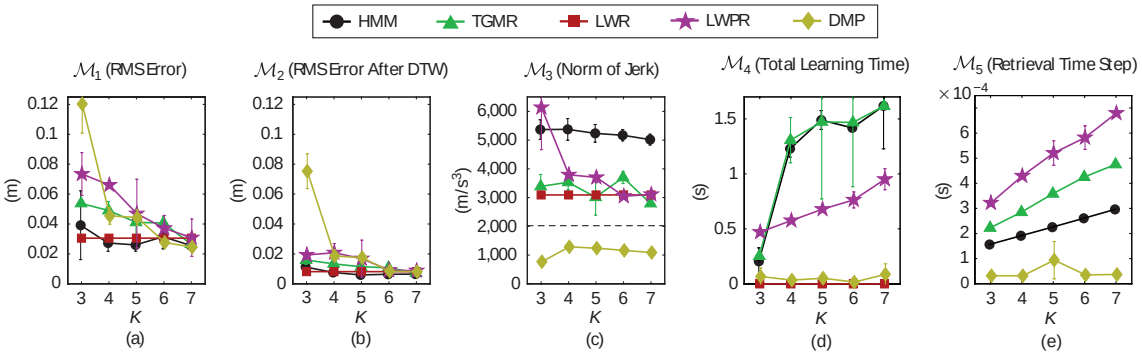

Figure 22: Quantitative comparison of several PbD approaches, depending on the number of states K. (Source: [27])

These diagrams are quite revealing. First of all, the investigated approaches are about equally accurate. Only DMP is moderately less accurate for a low number of states. The norm of jerk is an indicator of the smoothness of the trajectory. DMP produces the smoothest results (smoother than the original trajectory, depicted with a dashed line), HMM the least smooth trajectories. The batch learning time for both HMM and TGMR are relatively high, especially for higher K. While LWR does not need to learn in advance (zero learning time), its successor LWPR is about twice as fast as HMM/TGMR. DMP is significantly fast and remains fast even for a higher number of states. During execution, DMP again outperforms the other approaches. Nevertheless, all the other approaches stay below 1 ms and are therefore all usable

for real-time applications.

In [30], SEDS, GMR, LWPR and GPR (although unfortunately not in one of its latest variants LGP or LSOGP) are compared. The criteria are accuracy, learning time and retrieval time; the results are displayed in Table 2. The experiment performed was learning 20 human handwriting motions, which are shown in Figure 18.

Method	Average / Range of error $\overline{e}$	Average / Range of Training Time (sec)	Average Retrieval Time (sec)
SEDS-Likelihood	0.19 / [0.11 - 0.33]	19.23 / [2.22 - 48.26]	0.00068
GMR	0.13 / [0.08 - 0.28]	0.211 / [0.038 - 0.536]	0.00068
LWPR	0.17 / [0.07 - 0.35]	2.49 / [1.51 - 4.01]	0.00054
GPR	0.04 / [0.02 - 0.07]	52.42 / [22.84 - 142.27]	0.07809

Table 2: Quantitative comparison of the performance of different PbD approaches for learning handwriting motions. (Source: Adapted from [30])

In particular, this study adds insights about SEDS and basic GPR, which had not been compared in the aforementioned study. It can be seen that GPR is significantly more accurate than the other approaches, SEDS is slightly less accurate than GMR and LWPR. On the other hand, GPR is very computational demanding while training, by a factor of about 20 compared to LWPR and about 200 compared to GMR. In [27], the learning time of LWPR was shorter than for GMR. This shows how application and implementation dependent these figures are. For the retrieval time, SEDS is as fast as GMR. This is logical, as the complexity of the generated model is the same. GPR is much slower than the other methods, by a factor of around 100. GPR is therefore not real-time capable.

The competitiveness of a more recent implementation of GPR is evaluated in [23], which compares LGP with GPR. Here the accuracy of LGP is shown to be similar to basic GPR, while the retrieval time is sped up by a factor of about 5. While this is a significant improvement, the speed is still not as comparably fast as for the other approaches. For LSOGP no comparison was found.

2.5 Discussion

In the beginning of Section 2.4, the applicability of symbolic approaches (Section 2.2) and those utilizing spline encoding (Section 2.3.1) has already been discussed. Hence, the focus is here on statistical and dynamical approaches of PbD. Indeed each of these methods has its own unique features, but within the context of imitation learning, the number of competitive candidates can be narrowed down to two.

In the qualitative evaluation (Section 2.4.1), it was shown that both LWL and GPR have not been used much for PbD. The reason for this is that these approaches are more or less plain regression methods. In order to make them usable for imitation learning, they have to be equipped at least with a controller for the execution phase. This does not mean that LWL and GPR are useless, but they have to be seen as tools, especially for DMP.

Although SEDS has been considered here as its own approach (following the current literature), it can actually be classified together with GMR. The learning

method SEDS was developed for HMM without transition matrix, but it is probably possible to adapt it for HMM with transition matrix and for GMM. In that case, the advantages of SEDS and GMR are combined.

Thus for imitation learning, the choice should be between GMR and DMP. These have already been compared in literature [27, 30], and their similarities on the mathematical side were shown [35]. One major difference is the handling of time as described in Section 2.4.1. As commands from GMR (with HMM) depend only on the position, perturbations are handled inherently, while the time dependent DMP must be adapted with coupling terms to do so. On the one hand, this requires extra efforts, on the other hand, this also offers more flexibility. As already mentioned, the speed of the execution for DMP can simply be adjusted with a single time parameter. Another aspect of this is that GMR (with HMM) does not require to start the trajectory at the beginning, but is inherently able to execute only the "tail" of the trajectory. Using DMP, managing this situation requires logic to adapt the phase variable beforehand. Intrinsic to GMR is the equal treatment of discrete and cyclic motion, whereas for DMP this requires separate handling. This may seem disadvantageous, but according to [31], nature also distinguishes the two. An advantage of GMR is the learned correlation between different variables. In most DMP implementation, this is not considered, although there exists an approach to do so [11, 34]. Global stability is guaranteed in DMP with a decaying term, while GMR needs a special learning method such as SEDS, which is computational heavy. Computational efficiency is in general much higher for DMP than for GMR. Another aspect is that DMP generates an accurate model from just one demonstration, in contrast to GMR, which usually requires several. Finally, GMR (with transition matrix) can handle multiple trajectories in one model, which is not possible with DMP.

Summing up, both GMR and DMP are powerful PbD frameworks. Which one is preferable depends mainly on the application. When using GMR, the HMM with transition matrix seems to be advantageous (compared to GMM). In general, DMP might be the more flexible approach, as it allows interchangeable components and coupling terms for online modification of the trajectory.

2.6 Alternatives to Programming by Demonstration

A number of alternatives to PbD exist. These alternatives, including biologically-oriented approaches, direct programming and RL are briefly presented here, but a detailed analysis goes beyond the scope of this thesis.

Biologically-oriented learning approaches are significantly different from the other described methods. They are inspired by human and animal learning and make use of artificial neural networks imitating *mirror neuron system* (MNS) [36]. More information about MNS and their computational models can be found in [36] with a current revision in [37].

In contrast to direct programming, PbD offers the potential to teach new skills much faster. Direct programming is often tedious, error prone and sometimes might even be unfeasible, whereas PbD can be very intuitive for the teacher. Nevertheless,

direct programming is still widely used in the industry, where the focus is on accuracy and the environment is often highly predictable and customized for the specific needs.

Another alternative to PbD is RL. In the case of RL, the robot (or more in general the learning-system) learns fully autonomously using a reward function, which gives feedback about the correctness of the current performance. The system tries to maximize the reward function by trial and error [38]. RL offers some advantages over PbD [12]: First, tasks, which cannot be demonstrated, can still be learned. Second, tasks can be optimized according to a freely definable reward function. Third, tasks can be adapted to changed environments. On the other hand, several contemporary robot designs lean towards an increasing number of DoF, for which the search space is very big and computational demanding. In such cases, the probability of finding a global optimum can be negligible. However, reaching the global maximum is often not needed and neither reached by the other PbD methods.

Currently, RL receives an increasing role in PbD, as it can be used in combination with imitation learning: the task is first demonstrated to give an initial solution, which is then optimized using RL. One example for this approach is the pancake flipping robotic arm shown in Figure 23 [12].

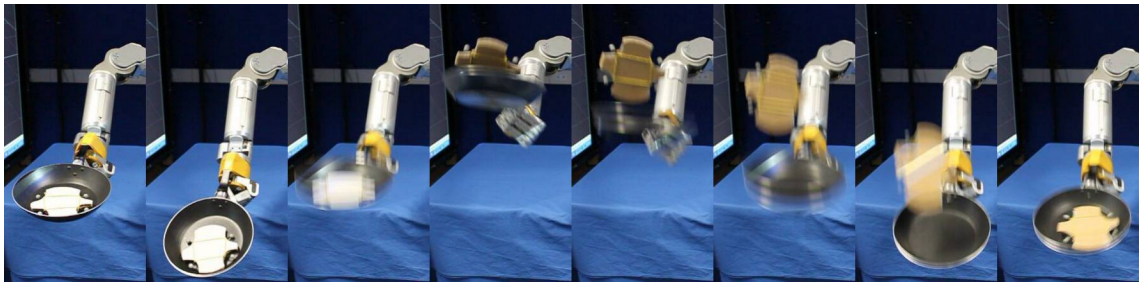

Figure 23: Image series of a robotic arm flipping a pancake imitation. (Source: [12])

In a long term perspective, [12] anticipates what they call “Goal-directed learning”. Here only the goal is shown, not how it is achieved. The computational complexity will increase, while the difficulties for the teacher decreases. This seems to put emphasis on symbolic approaches.

3 Dynamic Movement Primitives

The discussion in Section 2.5 concluded that, given the goal of in-contact tasks in dynamic environments, DMP is the most flexible framework. The capabilities are similar to GMR, but especially the coupling terms of DMP promise advanced dynamic modifications of the trajectory in the execution phase. That is why the DMP framework was chosen to be implemented and used for further research. Therefore, the details about this theoretical framework are presented in this chapter. First, the basics of DMP are presented. Subsequently, the fundamental equations are extended by novel algorithms for in-contact tasks.

3.1 Basics of Dynamic Movement Primitives

The basics of the DMP framework have already been introduced in Section 2.3.2 and its building blocks have been shown in Figure 14. The following section focuses on the details of the implementation of both the execution and the learning phase, in the case of a one dimensional task. Subsequently, the generalization to multiple dimensions is shown.

3.1.1 Execution phase

DMP have been introduced in the machine learning literature as a general and flexible framework. However, most implementations of DMP share a common mathematical formulation, which is presented in great detail in [33]. The implementations used in this thesis largely refer to these common implementations. To give a better impression of the meaning of the different equations, they are visualized for an exemplified 2 D system shown in Figure 24.

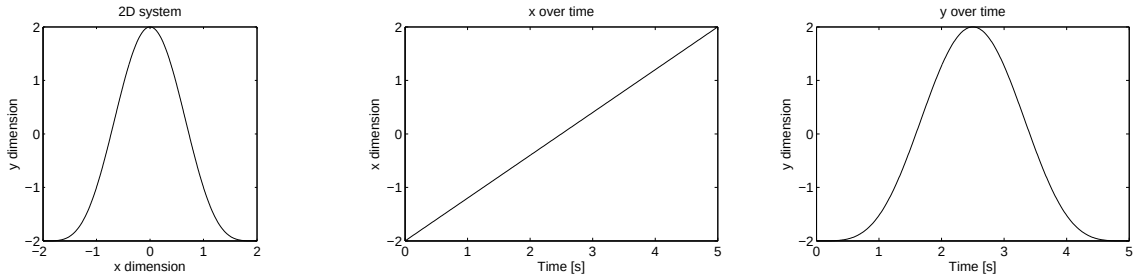

Figure 24: The plots show a 2 D system. The left plot displays the trajectory and the middle and right one give the single dimensions over time.

Transformation system The basic implementation of DMP considers each dimension of the system in isolation. Therefore, there is a 1 D transformation system for each dimension. As mentioned in Section 2.3.2, one crucial requirement for DMP is stability. For example, in the case of discrete (i. e. non-cyclic) movements, the system is supposed to asymptotically converge to the set point from any state. One

possible way to fulfill this requirement is by implementing the *transformation system* by the so called spring-damper-system

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

which can be written in first-order notation:

$$\begin{aligned} \tau\dot{z} &= \alpha_z(\beta_z(g - y) - z) + f(x) \\ \tau\dot{y} &= z, \end{aligned} \tag{2}$$

where τ is a time constant (temporal scaling factor), g is the goal (in the case of discrete motions, the point of asymptotic convergence) and y, $\dot{y}$, $\ddot{y}$ are position, velocity and acceleration (the output of the system). The positive parameters α_z and β_z are related to stiffness and damping. The system is critically damped for $\beta_z = \alpha_z/4$. The transformation is excited by the *force term* $f(x)$ in order to generate the desired trajectory, rather than an asymptotic approach to the set point.

To get a first impression, Figure 25 visualizes the effect of the interaction between the transformation system and the force term (in a 2 D system) as force field. This force field (shown as arrows) causes the system to move towards the current set point (i. e. the point all arrows are pointing to), at each time, independently on its current state. This is why the system is also called point attractor. The shift of the set point, dynamically changing in time, is caused by the force term.

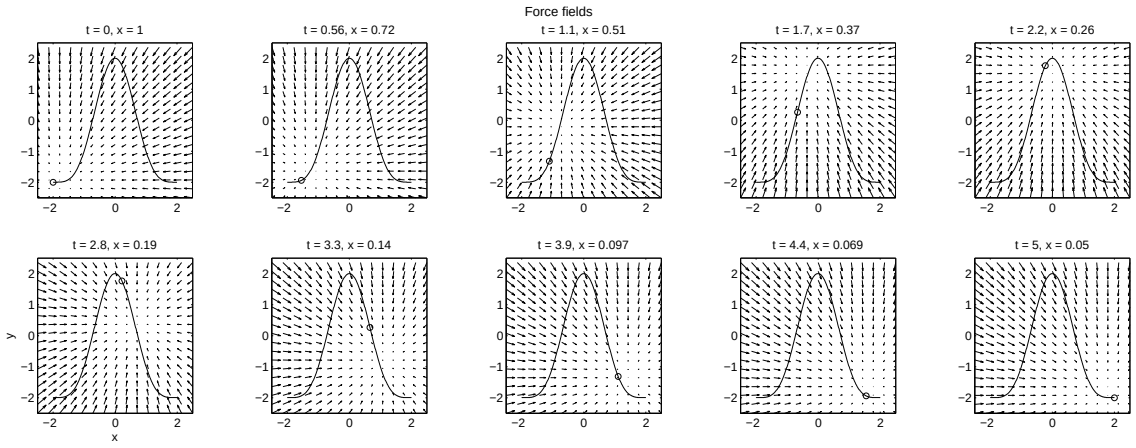

Figure 25: These graphs illustrate the force field generated by the transformation system (arrows), the overall trajectory (solid line) and current position (circle on the trajectory) at different time steps.

Function approximator It was shown that the force term is responsible for the system moving along the desired trajectory. This force term $f(x)$ is generated by a nonlinear *function approximator*. The function approximator takes a monotonically evolving variable x as input (the phase variable from the canonical system described next) and produces the appropriate force term, resulting in the anticipated trajectory.

One possibility for the approximator is the normalized linear combination basis function:

$$f(x) = \frac{\sum_{i=1}^{N_w} \Psi_i(x) w_i}{\sum_{i=1}^{N_w} \Psi_i(x)} \tag{3}$$

The basis functions are represented by $\Psi_i(t)$, which are weighted by w_i. N_w defines the number of basis function. A higher number of basis functions allows for better accuracy, but also causes higher computational time, both in the learning and execution phase. The choice of basis function offers several options, yet distribution functions are typically used. In our implementation, the basis function for discrete motions is the Gaussian function

$$\Psi_i(x) = \exp\left(-\frac{1}{2\sigma^2}(x - c_i)^2\right). \tag{4}$$

For cyclic motions, the basis functions are von Mises functions,

$$\Psi_i(x) = \exp\left(h(\cos(x - c_i) - 1)\right), \tag{5}$$

the circular variant of the Gaussian function. The choice of the centers c_i of the distributions, as well as the standard deviation σ and measure of concentration h will be explained in Section 3.1.2.

In the example in Figure 26 and 27, the number of basis functions is $N_w = 10$. Figure 26 shows the (non weighted) Gaussians $\Psi_i(x)$, both over time and phase variable. Figure 27 gives the corresponding weighting factors w_i for the two dimensions.

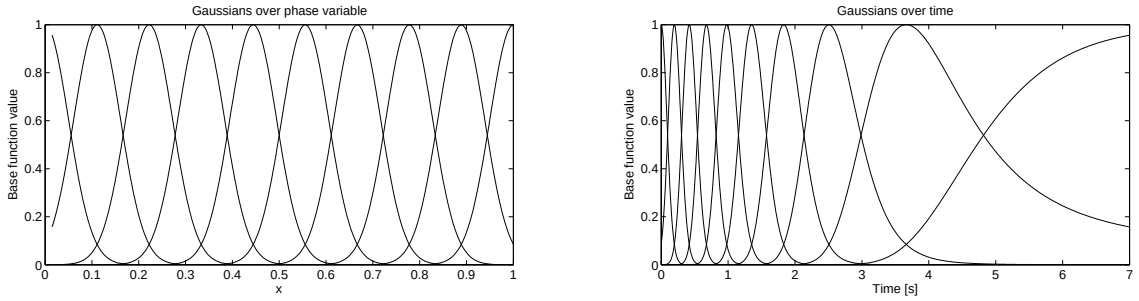

Figure 26: The ten Gaussian basis functions are plotted over time (right) and over the phase variable (left). The Gaussians are equally spread in the phase variable, not in time.

Canonical system The so called *canonical system* generates a phase variable x, which is used as input for the function approximator. The use of the canonical system allows the description of the evolution of the system in time without the explicit use of a time variable. Thus, the execution speed can be controlled, e. g. to take into account external perturbations (see Section 3.2.2).

As canonical system for discrete motions, the following first-order linear system is used:

$$\tau \dot{x} = -\alpha_x x. \tag{6}$$

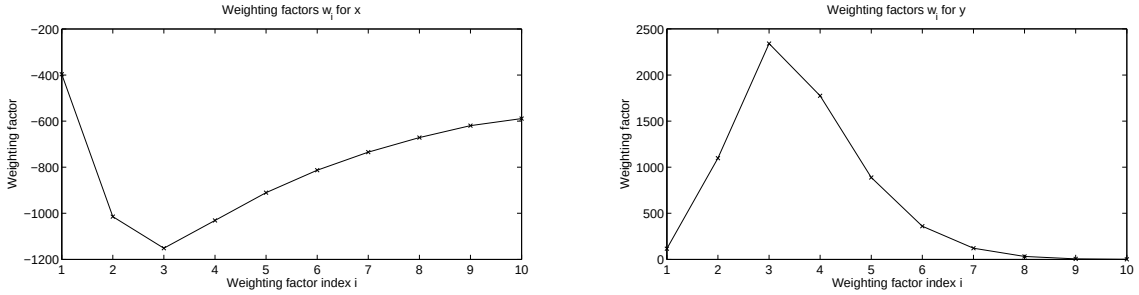

Figure 27: The ten weighting factors for the x dimension are shown in the left plot and those for the y dimension in the right plot.

The solution to this differential equation is (for constant τ and α_x)

$$x(t) = x_0 \exp\left(-\alpha_x \frac{t}{\tau}\right). \tag{7}$$

Here, τ is the temporal scaling factor from (3) and α_x the decay parameter. For rhythmic motions, the canonical system is described by the equation

$$\tau \dot{x} = 1, \tag{8}$$

with $x \in [0, 2\pi]$ yielding the solution

$$x(t) = \left(x_0 + \frac{t}{\tau}\right) \bmod 2\pi, \tag{9}$$

The initial value for the phase variable x for both systems is described by x_0, which is set to an arbitrary value (typically one for the discrete and zero for the cyclic system).

The canonical system generate a nonlinear relationship between the time t and the phase variable x. An example for the discrete canonical system described by (7) with $x_0 = 1$ and $\alpha_x = 3$ can be seen in Figure 28. The output of (9) is that of a saw-tooth generator.

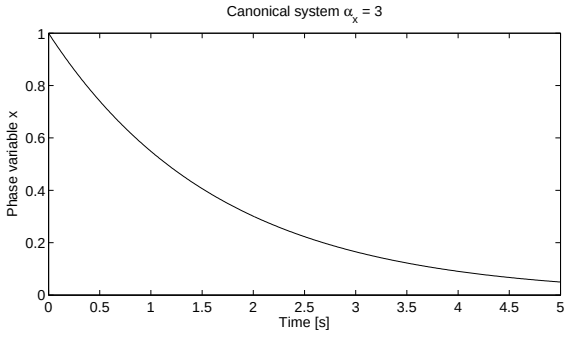

Figure 28: The exponential decay of the canonical system of a discrete system with $x_0 = 1$ and $\alpha_x = 3$.

Extensions To extend the possibilities of DMP and to add some helpful characteristics, two extensions are necessary [31, 33]:

First, the force term usually embeds a scaling term $\xi(x)$:

$$f(x) = \frac{\sum_{i=1}^{N_w} \Psi_i(x) w_i}{\sum_{i=1}^{N_w} \Psi_i(x)} \xi(x) \tag{10}$$

The basic scaling term, used e. g. in [33], is

$$\xi_{\text{basic}}(x) = (g - y_0)x \tag{11}$$

for discrete and

$$\xi_{\text{basic}}(x) = (g - y_0) \tag{12}$$

for cyclic motions (y_0 is the starting position). The phase variable x causes the force term of the discrete system to vanish as x approaches zero with increasing t. Therefore, the asymptotic stability of the transformation system is guaranteed[1]. This can be seen in the bottom right panel of Figure 25, where the current position coincides with the center of the force field, as the force term has approached zero and the system has reached its goal.

The term $(g - y_0)$ ensures the invariant properties of the attractor landscape when spatially scaling the motion [33]. This is further explained and proved in [33]. Yet, this term is problematic for motions having a starting and end point very close nearby. That is the reason why $(g - y_0)$ in both (11) and (12) is replaced by $A_{\max} = \max(y) - \min(y)$, as also suggested by [33]. $A_{\max}$ is the maximum amplitude of the trajectory, as $\max(y)$ is the maximum and $\min(y)$ the minimum value of all trajectory points y. This yields

$$\xi_{\text{minmax}}(x) = A_{\max} x = (\max(y) - \min(y))x \tag{13}$$

for discrete and

$$\xi_{\text{minmax}}(x) = A_{\max} = \max(y) - \min(y) \tag{14}$$

for cyclic motions.

The second extension is the inclusion of *coupling terms* in the transformation and canonical system. These coupling terms are intended for online modulation of the trajectory. As they are only used during the execution phase, the coupling terms are added to the differential equations. The *temporal coupling term* C_c is added to the canonical system, thus equation (6) and (8) are changed to

$$\tau \dot{x} = -\alpha_x x + C_c \tag{15}$$

respectively

$$\tau \dot{x} = 1 + C_c. \tag{16}$$

[1]Similarly, BIBO stability can be proved for cyclic movements [33]

Similarly, the *spatial coupling term* C_t is added to the transformation system, causing (2) to become

$$\begin{aligned} \tau \dot{z} &= \alpha_z(\beta_z(g - y) - z) + f(x) + C_t, \\ \tau \dot{y} &= z. \end{aligned} \tag{17}$$

One implementation for the temporal coupling term is shown in Section 3.2.2, which uses C_c to handle perturbations during the execution of the movement. The spatial coupling term is not used in this thesis, yet an example can be found in [33], which exploits C_t to avoid obstacles (see also Figure 21).

3.1.2 Learning phase

For the determination of the different parameters, a variety of methods exist. One common choice is LWR [33], which is described in this section.

Input The input of the learning algorithm is a certain number N_D of demonstrations (one is sufficient), which are indexed by $j = [1 \dots N_D]$. The trajectory of every demonstration consists of sets of time t^j, position y^j, velocity $\dot{y}^j$ and acceleration $\ddot{y}^j$, each of size M_j. In the case that the velocity and acceleration information are not given, $\dot{y}^j$ and $\ddot{y}^j$ have to calculated from t^j and y^j.

Trimming When recording a trajectory, the recorder is first manually started, then the robot is moved and finally the recorder is manually stopped. Therefore, the robot is typically not moving at the beginning and end of the recorded sets. In order to reduce the representation to the essential moving part of the demonstration, the sets should be trimmed. This is done according to [33]: the maximum velocity $\dot{y}^j_{\max} = \max(\dot{y}^j_k, k \in [1 \dots M_j])$ is determined and all samples at the beginning and end of the sets j are cut off, for which the velocity is lower than $0.02 \cdot \dot{y}^j_{\max}$. The number of elements of each set of a demonstration is now M'_j with $M'_j \leq M_j$.

Learning With the input data having been trimmed, for each demonstration j the following parameters and values are calculated:

- The goal is given by
$$g^j = y^j_{M'_j}. \tag{18}$$
- The time factor τ, which is simply the duration of the demonstration can be derived from
$$\tau_j = t^j_{M'_j} - t^j_1. \tag{19}$$
- For the next step, a set f^j of forces is calculated based on (1)
$$f^j_k = \tau_j^2 \ddot{y}^j_k - \alpha_z(\beta_z(g^j - y^j_k) - \tau_j \dot{y}^j_k). \tag{20}$$

This leads to

$$J_i = \sum_{k=1}^{M'} \Psi_i(x(t_k)) \left(f_k - w_i \xi(x(t_k))\right)^2, \tag{21}$$

which is a weighted linear least square problem. To find the weighting factors w_i using LWR, the vectors $\mathbf{s}$ and $\mathbf{f}$ and matrices $\mathbf{\Gamma}_i$ first have to be set up:

$$\mathbf{s}^j = \begin{pmatrix} \xi_1^j(x(t_1)) \\ \xi_2^j(x(t_2)) \\ \vdots \\ \xi_{M'_j}^j(x(t_{M'_j})) \end{pmatrix} \qquad \mathbf{s} = \begin{pmatrix} \mathbf{s}^1 \\ \mathbf{s}^2 \\ \vdots \\ \mathbf{s}^{N_D} \end{pmatrix} \tag{22}$$

$$\mathbf{f}^j = \begin{pmatrix} f_1^j \\ f_2^j \\ \vdots \\ f_{M'_j}^j \end{pmatrix}, \qquad \mathbf{f} = \begin{pmatrix} \mathbf{f}^1 \\ \mathbf{f}^2 \\ \vdots \\ \mathbf{f}^{N_D} \end{pmatrix} \tag{23}$$

$$\mathbf{\Gamma}_i^j = \begin{pmatrix} \Psi_{1,i}^j(x(t_1)) & & & 0 \\ & \Psi_{2,i}^j(x(t_2)) & & \\ & & \ddots & \\ 0 & & & \Psi_{M'_j,i}^j(x(t_{M'_j})) \end{pmatrix} \quad \mathbf{\Gamma}_i = \begin{pmatrix} \mathbf{\Gamma}_i^1 & & & 0 \\ & \mathbf{\Gamma}_i^2 & & \\ & & \ddots & \\ 0 & & & \mathbf{\Gamma}_i^{N_D} \end{pmatrix} \tag{24}$$

$\xi_k^j(x)$ is the aforementioned scaling term and $\Psi_{k,i}^j(x)$ the basis function. The weighting terms w_i are then calculated as follows:

$$w_i = \frac{\mathbf{s}^T \mathbf{\Gamma}_i \mathbf{f}}{\mathbf{s}^T \mathbf{\Gamma}_i \mathbf{s}} \tag{25}$$

Only a few parameters remain. The number N_w of basis functions must be predefined based on the demonstrations and the desired accuracy. For discrete motions, the centers c_i of the Gaussian functions are spread equally between zero and one

$$c_i = \frac{i-1}{N_w - 1} \tag{26}$$

and the standard deviation is set to $\sigma = 0.5/N_w$, which was determined experimentally. For cyclic motions, the centers of the von Mises functions are accordingly spread between 0 and 2π

$$c_i = 2\pi \frac{i-1}{N_w - 1} \tag{27}$$

and the measure of concentration is set to $h = 4N_w^2$ ($1/h$ is analogous to σ^2). The decay parameter $\alpha_x > 1$ should be choosen low (< 2) for accurate trajectory following and high (> 5) for a precise goal approach.

3.1.3 Multiple dimensions

A trajectory in an N-dimensional space is represented by $N_D \geq N$ components. For example, in a 3 D space, there can be three components for the position, three for the orientation (or four when using quaternions as representation) or one for each joint when working on the joint level.

DMP handle multiple components easily. Each component has its own transformation system and function approximator. Consequently, for each component, the weighting factors w_i are learned independently. However, the parameters α_x, α_z, β_z, τ, N_w, c_i, σ and h usually coincide.

In the execution phase, the function approximator and transformation system are also calculated independently. The connecting and synchronizing part is the unique canonical system, which is shared by all dimensions.

3.2 Dynamic Movement Primitives for in-contact tasks

Motions for the execution of in-contact tasks require several extensions and adaptations to basic DMP implementations. These have been developed in this thesis and to our knowledge, they present novel approaches. When the robot is in contact with another object, two different situations must be distinguished and appropriately handled: (i) intended contacts and (ii) not intended contacts. The former occurs when the robot was taught to interact with the environment. The latter can take place at any time, when a robot accidentally collides with objects in its (dynamic) environment and can be considered as a perturbation. The handling of both cases is heavily based on the capabilities of the controller, which is described in the following section. After that, the developed algorithms for perturbations and in-contact tasks are examined.

3.2.1 Controller

In the execution phase of DMP, a trajectory is generated by the transformation system. This trajectory can be on the joint level, Cartesian level and can also contain desired forces (see Section 3.2.3). Subsequently, one or more controllers are responsible for converting the trajectory to motor commands (see Figure 14).

The controllers used for the implementation are a Cartesian and a joint impedance controller. Next to the set point or equilibrium point (the desired position), also the stiffness and damping factors can be defined. Either on joint or Cartesian level, the controller then imitates a spring-damper-system. Additionally, also an exerted F/T can be defined, which is added to the spring-damper-system. The controllers have a very accurate dynamical model of the arm, which allow for automatic compensation of dynamical forces such as gravity or Coriolis force.

The algorithms developed in this thesis can be applied to any hardware with controllers equivalent to the described ones. The *KUKA Robot Controller* (KRC) (see Section 4.2) of the KUKA LWR, which is used as hardware in this thesis, offers those two controllers. For stiffness and damping, only the diagonal values of the corresponding matrix can be set, not the correlation factors. It can also be decided,

whether the rotational frame of reference is fixed in the base or should be relative to the current tool orientation. The control law for the controllers is described in [39]. For the joint impedance controller, it is

$$\tau_{\text{cmd}} = k_j(q_{\text{des}} - q_{\text{msr}}) + D(d_j) + \tau_{\text{des}} + f_{\text{dyn}}(q, \dot{q}, \ddot{q}). \tag{28}$$

For the Cartesian impedance controller, it is

$$\tau_{\text{cmd}} = J^T(k_c(x_{\text{des}} - x_{\text{msr}}) + D(d_c) + f_{\text{des}}) + f_{\text{dyn}}(q, \dot{q}, \ddot{q}). \tag{29}$$

Here, q_{des} and q_{msr} are the desired respectively measured joint positions, equivalently x_{des} and x_{msr} for the Cartesian pose (position and orientation). The stiffness and normalized damping parameters are k_j/k_c and d_j/d_c for the joint/Cartesian level. The desired joint torques are expressed by τ_{des} and the desired Cartesian F/T by f_{des}. The Cartesian command is converted to joint torques τ_{cmd} by the transposed Jacobian J^T. Finally the dynamics of the arm are taken into account by the term $f_{\text{dyn}}(q, \dot{q}, \ddot{q})$. More details about the two controllers can be found in [40].

When not working with forces, the trajectory y generated by the transformation system is simply forwarded to either of the two impedance controllers ($q_{\text{des}}/x_{\text{des}}$) with the desired F/T set to zero ($\tau_{\text{des}}/f_{\text{des}}$). The stiffness α_z and damping values β_z of the spring-damper transformation system of the DMP are being assigned to the same values as used for the spring-damper-system of the controller, as this resulted in the most accurate trajectory reproductions.

3.2.2 Handling of perturbations

As one of the aims of this thesis is a robot safely interacting with humans, much care has to be taken to prevent harmful collisions. Although it would be preferable to always avoid undesired collisions, this is currently not possible. That would require external sensors, for example a camera.

Therefore, some precautions must be made to reduce the risks of a collision. First of all, the smooth and rounded design of the arm (see Section 4.2) is a big step towards this. Although this design is already biologically inspired, the construction is still very stiff compared to soft organic tissues.

The previously described impedance controller further mitigates the impact of a collision for flexible objects, as the arm is compliant to a certain extent. On removal of the obstacles, the arm asymptotically approaches the desired set point.

The KRC arm has an additional feature, which allows to restrict the maximum F/T exerted on the environment. This must of course make use of the torque sensors and the dynamic model of the arm, to distinguish between internal forces (gravity, Coriolis, friction, acceleration) and external forces (collisions). The maximum allowed F/T can be set for each joint or Cartesian dimension. When the calculated external F/T is higher than the allowed one, the arm stops moving and even gives back. Torques are measured in a 3 kHz cycle [40] for each joint, thus reactions to collisions are fast.

Nonetheless, also the DMP algorithm has to be adapted to stop or slow down the execution in case of perturbations. In case this precaution is omitted, removing an object after the robot had stopped would cause a fast and direct movement to the set point of the unperturbed trajectory. One way of coping with perturbations was described in [33]. There, the spatial coupling term C_t (see Section 3.1.1) is set proportional to the absolute error (which is low-pass filtered) and τ is set to a value proportional to the squared error. There are several problems with this approach. First, it does not just establish a coupling term, but also changes the parameter τ. Second, new non-intuitive parameters have to be introduced and tuned manually. Third, it is not clear how to implement the algorithm for a multi-dimensional case.

Because of these limitations, a novel approach to manage disturbances has been developed in this thesis. The targets targets of this approach were the ability to handle several dimensions, an easy parametrization and an implementation using only the temporal coupling term C_c. To achieve this, initially a maximum deviation $\Delta e^j_{\max}$ from the desired trajectory for all dimensions $j \in \{1, \ldots, N_D\}$ is defined, for which the algorithm shall cause a full stop of the execution progress. This can for example be a fixed length, be relative to the maximum amplitude of the dimension or the maximum of the two previous options:

$$\Delta e^j_{\max} = \max(5\,\mathrm{cm}, 0.05 A^j_{\max}). \tag{30}$$

In each iteration, for every dimension, the current error

$$\Delta e^j = y^j - y^j_{\text{measured}} \tag{31}$$

is determined and the absolute value of the relative error calculated as

$$e^j_{\text{rel}} = \left| \frac{\Delta e^j}{\Delta e^j_{\max}} \right|. \tag{32}$$

The maximum $e_{\max,\,\text{rel}}$ of the relative errors of all dimensions determined

$$e_{\max,\,\text{rel}} = \max(e^j_{\text{rel}} \mid j \in \{1, \ldots, N_D\}) \tag{33}$$

and limited to one

$$e'_{\max,\,\text{rel}} = \min(1, e_{\max,\,\text{rel}}) \tag{34}$$

To stop the canonical system (i. e. $\dot{x} = 0$), the temporal coupling term would have to be set to

$$C_{c,\,\text{stop}} = \alpha_x x \tag{35}$$

for discrete motions and

$$C_{c,\,\text{stop}} = -1 \tag{36}$$

for rhythmic motions (see Equations (6) and (8)). By assigning

$$C_c = C_{c,\,\text{stop}} e'_{\max,\,\text{rel}}, \tag{37}$$

the canonical system is slowed down when deviating from the desired trajectory and finally stopped, when the error $e_{\max,\,\text{rel}}$ grows above one. After the error reduces eventually, the canonical system continuous. Thus, the canonical system adjusts not just to external perturbations, but also to non-feasible trajectories.

3.2.3 Handling of desired forces

In-contact tasks require another extension of the basic DMP algorithm, especially on the controller side (the part after the transformation system). For the experiments described in this thesis, the implementation limits exerted forces to the Z direction (in the tool frame) and when working under the Cartesian impedance controller. Forces in that direction are probably the most common ones (e. g. pressing a button, writing with a pen).

If the robot is supposed to exert forces on other objects, also the force profile is needed in the demonstration. Therefore, additionally to the pose, also forces in the Z direction are recorded using the F/T-sensor. In the learning phase (see Figure 34), the force is just treated as an additional dimension.

For the execution, two additional force controllers were designed, which work on top of the Cartesian impedance controller. The first is responsible for the approaching velocity, the second one for the exerted force. A state machine of these controllers is shown in Figure 29. If during the execution the desired force f_{DMP} (i. e. y of the transformation system for the force component) is above a certain threshold $f_{\mathrm{des,\,min}}$, the stiffness for the Z-dimension is set to zero. This has the effect that the position in Z-direction is no longer controlled by the spring-damper system. Movements in Z-direction are then only generated by the force term f_{des} in equation (29). This follows the suggestions from [3].

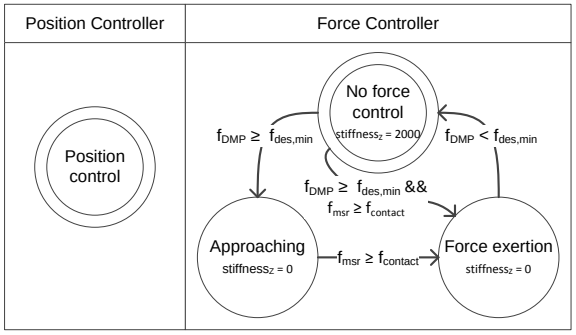

Figure 29: State machine for in-contact tasks. The position and force controller run in parallel. The approach state is activated when $f_{\mathrm{DMP}} \geq f_{\mathrm{des,\,min}}$, the force exertion starts with $f_{\mathrm{msr}} \geq f_{\mathrm{contact}}$ and stops again when $f_{\mathrm{DMP}} < f_{\mathrm{des,\,min}}$.

The use of the velocity controller allows an object to be displaced in positive Z-direction compared to its position during teaching. As long as there is no contact ($f_{\mathrm{msr}} < f_{\mathrm{contact}}$) but $f_{\mathrm{DMP}} > f_{\mathrm{des,\,min}}$ holds true, the tool shall be moved in Z-direction with a fixed speed v_{approach} by applying a force f_{des}. A *proportional-intergral-derivative controller* (PID controller) is used, which compares the current speed v_{cur} (difference in position divided by elapsed time) to v_{approach} and adapts f_{des} appropriately[2].

[2]The gain of a PID controller is the sum of the weighted error ($K_P \cdot e(t)$), weighted sum of the error over time ($K_I \cdot \int e(t)\Delta t$) and a weighted change in error since the last cycle ($K_D \cdot \Delta e(t)/\Delta t$) [41]

The force controller is needed due to the limitations of the force estimation of the LWR arm. Although much care has been taken to model the payload data (see Section 4.2) and to calibrate the internal torque sensor, the offset of the LWR force measurements could not be decreased below about $\pm 1\,\mathrm{N}$. This is a mainly pose dependent offset or bias (a systematic error), which does not affect relative measurements or commands. Here again, a PID controller was used that compares f_{DMP} with the F/T-sensor measurement f_{msr} to adjust f_{des}.

One problem occurs in case the contact is lost ($f_{\mathrm{DMP}} < f_{\mathrm{des,\,min}}$). When this occurs, the stiffness is being reset. The arm would consequently jump to the point where it would have lost contact with a non-displaced object. As this is of course not desired, precautions must be taken. The developed solution is rather straightforward. At the switching point, the offset between the current pose and the desired pose is determined ($\Delta x = x_{\mathrm{msr}} - x_{\mathrm{DMP}}$) and added to all future pose commands: $x_{\mathrm{des}} = x_{\mathrm{DMP}} + \Delta x$.

Some further adaptions are made to the DMP algorithm to ensure a smooth execution:

- Once the force controller is activated, it is not allowed to switch back to the velocity controller before the the contact is intentionally abandoned ($f_{\mathrm{DMP}} < f_{\mathrm{des,\,min}}$, see Figure 29). This prevents oscillations resulting from periodical switches between the two controllers at the moment the contact is made.

- As was shown in Section 3.2.2 and will later be further explained (Section 4.6.4), the controllers introduced in this section can influence the temporal coupling term by adjusting the feedback to the DMP perturbation handler. This is used in two ways. First, while approaching an object, the feedback of the force measurement is altered and set to the value causing the canonical system to stop (see Equation (35)). The idea here is that a displaced object is similar to a perturbation of the trajectory, just the reference frame is different. Second, while in contact, this force measurement feedback is again changed and set to the desired value f_{DMP}. This prevents force errors from having an effect on the canonical system. The reason here is the relatively high force error, which would prevent a continuous execution when being in-contact. When in contact, also the feedback of the measured pose is set equivalently to the desired one. This must be done to ignore the positional offset of the surface.

4 Infrastructure

For the implementation and execution of the algorithms, a sophisticated hardware and software environment is needed. The infrastructure used in this thesis, capable of hard real-time robot sensing and control, is described in the following sections.

4.1 Overview

A general overview showing both hardware and software is given in Figure 30. It shows the three different hardware systems (external computer, KRC and arm) and how they are made up on hardware and software level. The operating system of the external computer with its real-time extension and real-time Ethernet drivers had to be set up and ad hoc configured (Section 4.2). Although this is to date a non trivial technical task, the development of the thesis was mainly focused on the components highlighted in green, ,amely, the program running on the KRC (Section 4.4) and the *Robot Operating System* (ROS) packages as well as the *Open Robot Control Software* (Orocos) components for the external computer (Section 4.5).

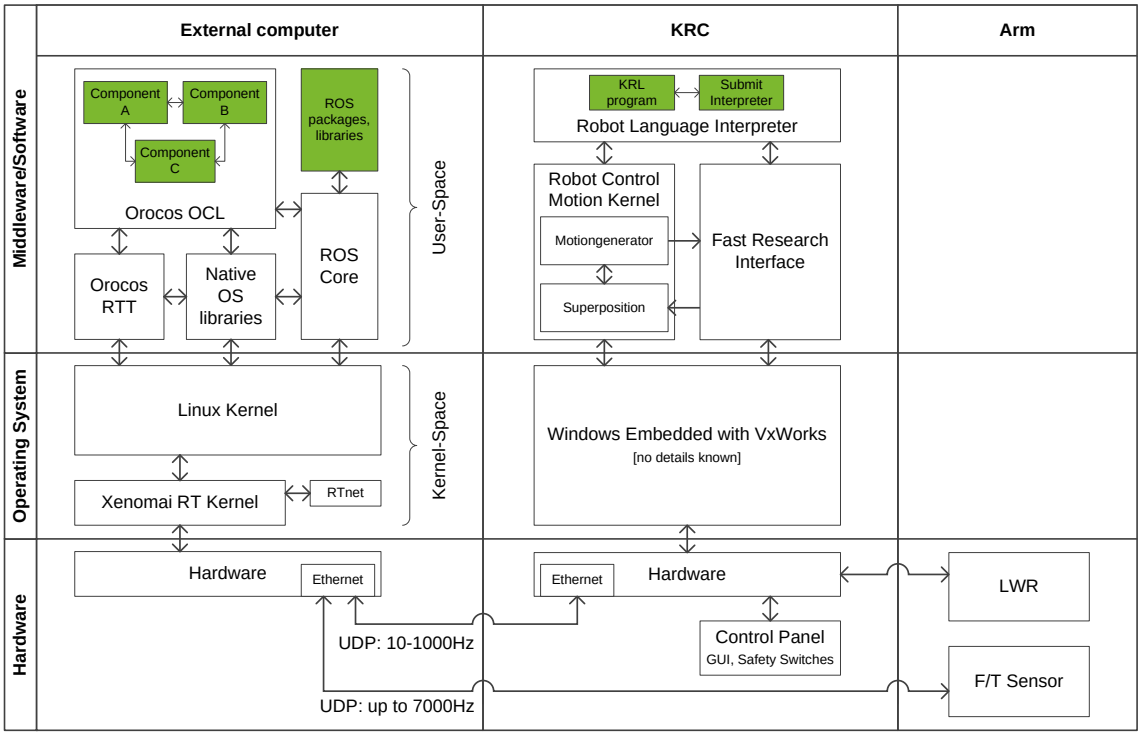

Figure 30: System overview. The infrastructure consists of the external computer, the KRC and the LWR with the F/T-sensor. Only the green components were developed in the thesis. The white components represent either commercial hardware or existing software (mainly operating system and middleware). (Sources: [39, 42, 43, 44])

4.2 Hardware

Industrial robotic arms are not comparable to human arms. They may be highly accurate, fast and offer redundant degrees of freedom. Nevertheless, they typically lack sophisticated sensors to explore the environment. Most robotic arms available only have encoders as sensors, so they can only determine the current position of each joint. Some of these arms are backdriveable, meaning that the rotational joints are compliant and can be moved by hand. This allows kinesthetic teaching. However, newly developed arms, such as the KUKA LWR used for this research, are equipped with integrated torque sensors in each joint. This allows them to precisely measure F/T acting on the arm.

While this is a huge improvement, allowing much more sophisticated applications [2], these arms still do not reach the multimodal sensitivity of human arms. They still lack a kind of skin, providing fine-grained information about pressure, temperature, vibration, touch, strain and more. These senses belong to the somatosensory system [45] and are essential for human movements and interactions [46].

For the experimental part of this thesis, the KUKA LWR 4+, a robotic arm with 7 DoF was used. The arm was given for this thesis, but proved to be a suitable platform for PbD experiments. Albeit it is a cutting-edge robotic arm with highly accurate torque sensors in each joint, it is still outperformed by the capabilities of a human arm as previously described. However, it allows state-of-the-art research for PbD and in-contact tasks. The dynamic model of the arm is known, which allows sensitive measurements of external forces.

The KUKA LWR 4+ was designed with research in mind [13]. With a mass of 16 kg, it is a lightweight arm having a rated payload of 7 kg. Its repeatability is ± 0.05 mm and the maximum outreach is about 0.8 m [47]. Thanks to its round shape with no sharp edges, in combination with the torque sensors and the dynamic model, the robot is safe to use for interactions with humans [40]. What is crucial for the development of this thesis on this platform is its active compliance: stiffness and damping can be dynamically changed on joint or Cartesian level.

The arm is connected to the so called KRC, an industrial computer controlling the arm. It is coupled with the *KUKA Control Panel* (KCP) that consists of a screen, a keyboard with additional keys and a 6 D mouse (see Figure 31). The KRC can communicate with an external computer using a *User Datagram Protocol* (UDP) Ethernet connection. In this case, the external computer runs the main program.

In the setting used in this thesis, a 6 D F/T-sensor[3] is installed between the tip of the arm and the tool, which was also given beforehand. This sensor allows to measure the applied F/T during demonstration and enables feedback measurements in the execution phase. The maximum allowed force is 290 N in X/Y and 580 N in Z-directions. For torques, the limit is 10 Nm. The force resolution is 1/8 N and the torque resolution 1/376 Nm and 1/752 Nm for X/Y and Z-axis respectively [48]. The sensor can provide measurements with a rate of up to 7000 Hz and allows the activation of different low-pass filters [49]. It is directly connected to the external computer using a second UDP Ethernet connection.

[3] ATI Mini45 F/T Transducer

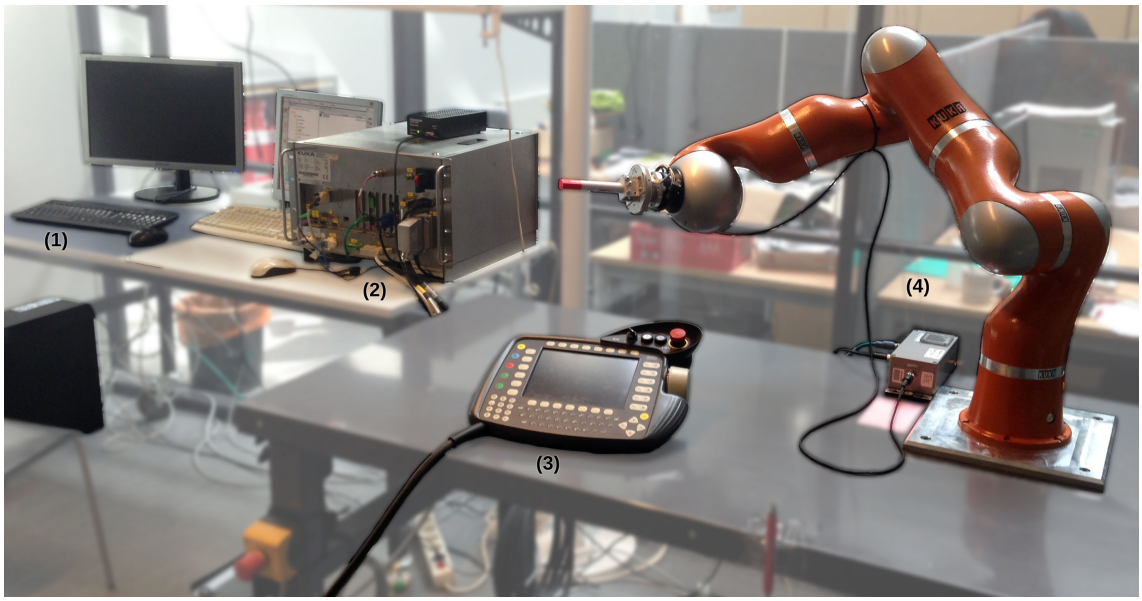

Figure 31: Hardware overview. The LWR arm (4) and the KCP (3) are both connected to the KRC (2). The KRC is via Ethernet connected to the external computer (1). The arm is equipped with a F/T-sensor and a pen tool.

An ad hoc tool has been designed and built for this thesis and mounted on the robot flange. The full tool assembly is shown in Figure 32. The flange cover mainly serves as protection for the cables coming out of the tip of the arm. The F/T-sensor is mounted on an adapter, which is fixed with screws and several plastic distance washers to the flange. This is done for two reasons. On the one hand, it admits air ventilation, thus allowing optimal cooling of the robot motors. On the other hand, it insulates the F/T-sensor thermally, limiting undesired thermal effects. To allow easy and fast exchange of different tools, a tool plate has been designed, which is joined to the assembly by a big threaded ring. In Figure 32, a custom pen holder is mounted on this tool plate, which is used for some of the experiments in chapter 5. The two handles have been integrated to ease the manual movement of the arm.

The KRC needs exact information about the tool mass, center of gravity and moments of inertia. This permits the activation of the gravity compensation mode, but also improve vibration damping. Therefore, all parts of the tool were recreated in a CAD program[4] with respective material and masses. Using the geometrical and physical properties, the program is able to calculate the necessary physical data.

The external computer provided for the thises is a commercial eight core desktop computer with two additional network cards[5] for the UDP Ethernet connections. The Linux operating system[6] is extended with a Xenomai real-time kernel[7]. Also the

[4]Autodesk Inventor Professional 2015
[5]Intel 82541PI Gigabit Ethernet Controller
[6]Ubuntu 12.04 LTE
[7]Version 2.6.2.1

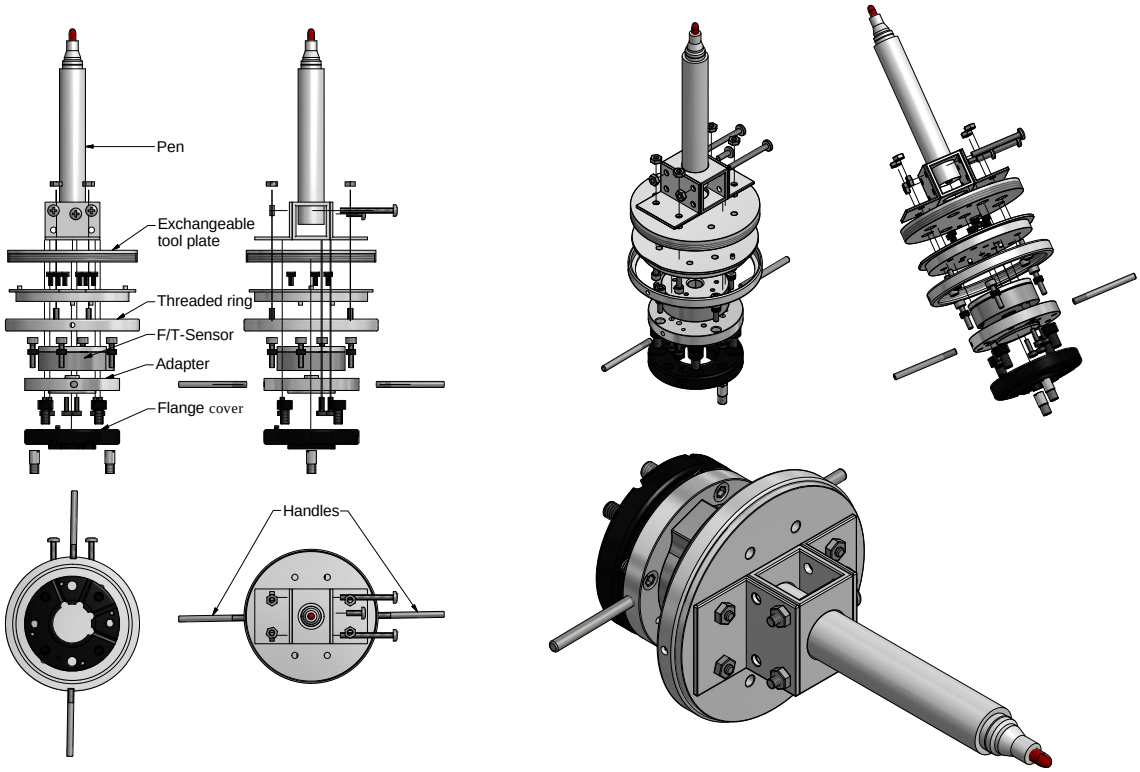

Figure 32: CAD explosion views from the pen tool with an assembled view in the lower right corner.

network cards use real-time drivers[8], in-order to meet the hard real-time requirements of the communication with the KRC and the F/T-sensor.

4.3 Fast Research Interface

In order to control the arm from an external computer, a new interface was developed by a cooperation between DLR[9] and KUKA, namely the *Fast Research Interface* (FRI) [39]. The FRI implements a communication protocol for a UDP Ethernet connection between the KRC and an external computer, working at a fixed rate of up to 1 kHz.

The communication supports two states. While in monitor state, data packages are sent from the KRC to the external computer but do not have to be acknowledged. Thus the current status of the robot can be observed, but not changed. The command state can only be activated, when the external computer is able to prove excellent quality of communication by replying to all packages within the previously defined rate. In that state, the external computer can command the arm, configure the impedance (stiffness and damping), set the desired position in joint or Cartesian space and determine the F/T the arm exerts on the environment. The messages

[8] RTnet 0.9.13

[9] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR, *German Aerospace Center*), Köln, Germany

from the KRC contain the measured and commanded positions, the measured F/T, as well as status information about communication, hardware and software.

Both messages from and to the external computer contain each 16 boolean, integer and double typed variables, which can be freely used for communication. They are intended to send additional values over FRI that can extend the functionality. Those variables can also be linked to inputs and outputs of the KRC and thus drive external industrial controllers or read values from them.

4.4 KUKA Robot Language

The *KUKA Robot Language* (KRL) is the programming language for programs running on the KRC. The language is quite simple in its syntax and shares many similarities with Pascal. It is the only possibility to directly program the KRC. It is used here to extend the capabilities of the FRI. For example, there is no native way to switch between monitor and control state using only the FRI. However, utilizing the variables of the FRI as flags and a custom KRL program, this can be realized.

To execute commands parallel to long lasting tasks (especially point-to-point and linear movements), a so called *submit interpreter* runs in the background. It is a special KRL program, which is called every 12 ms and therefore should run only for a very short period of time. As part of the background logic of the KRC, it is independent of the user program.

4.5 Middleware: ROS and Orocos

To write complex programs without having to start from the very beginning, usually some kind of middleware software is used. For robotic applications, a huge variety of such frameworks is existing. We are using Orocos (version 2.6) [50] for software parts requiring real-time control and ROS (Groovy) [51] for all other parts.

Orocos is under open-source license and was developed as multi-purpose control software for robots [52]. Being modular and flexible, it is supposed to run on all kinds of platforms. Orocos has a real-time motion control core with event-triggered interactions [53]. This real-time capability is the main reason for choosing Orocos, as the communication with the KRC requires hard real-time responses up to 1 kHz. Orocos programs are organized in components. Having been initialized, components are either activated periodically or on external events. After activation, the component has to yield quickly, in order not to violate the real-time restrictions. Communication between components utilizes so called ports, which have a fixed type. A component writes values to its output ports and reads from its input ports, which have to be connected to other component's output ports. Components can also offer operations or call operations from other components. See Figure 33 for a more detailed graphical overview.

ROS is also open-source and arguably has become the de-facto standard robotics middleware [54]. It was therefore chosen as middleware for non real-time critical tasks in this thesis. Programs in ROS consist of nodes, which can run on different machines that are connected in a network [55]. These nodes communicate using

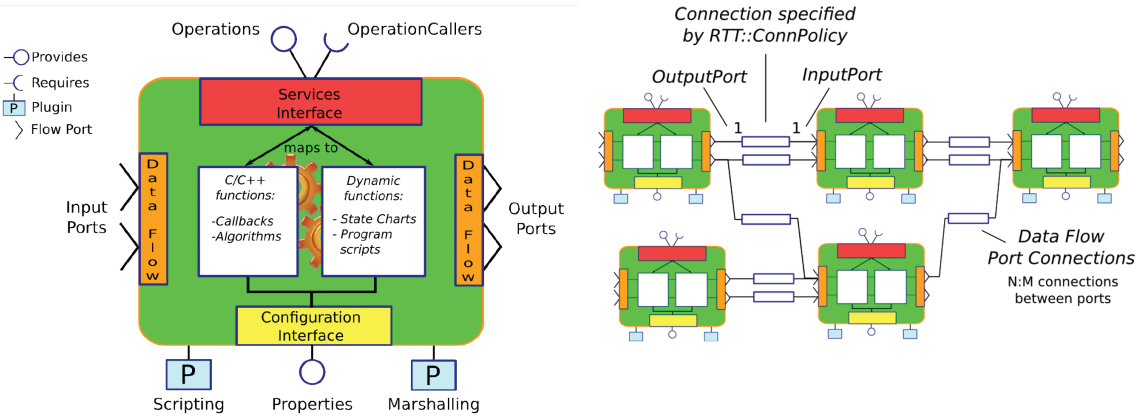

Figure 33: Orocos components mainly interact using ports and operations (left). Ports are connected to other component's ports (right). (Source: [43])

messages targeted to a certain topic having a distinctive type. A huge number of packages have been developed for ROS to extend its capabilities, allowing a fast development for all kinds of robots [56]. However, ROS does not meet real-time requirements.

Luckily, Orocos has the possibility to share messages with ROS. This interaction is limited[10], but sufficient for our purposes. Output ports in Orocos can be set to send their values as messages to ROS or vice-versa receive values from messages on input ports. In the context of this thesis, serialization is used to exchange whole objects between the two frameworks (see Section 4.6.3).

4.6 Software architecture

The sequence of steps involved in learning a trajectory is simple and shown in Figure 34. The process can be divided into three phases. During the recording phase, one or more users demonstrate the trajectory at least once, which is recorded in separate files for offline processing. Then, in the learning phase, an algorithm calculates the necessary parameters and stores them in another file. Finally, the trajectory is replayed utilizing the parameters generated during the execution phase.

Generally speaking, the developed software architecture implements the components from Figure 34. However, not only the DMP algorithms had to be integrated. As few code was available for the LWR, also a basic framework with a higher-level interface had to be developed. The programming language used is entirely C++(11), except the program running on the KRC, which is written in KRL. A class diagram including the crucial components can be found in Appendix A.

The DMP framework is modular and flexible, as was shown in Section 2.5. The software architecture was designed in a way to reflect these attributes. Therefore, logical units are encapsuled in packages and components. Classes are assigned to hierarchical namespaces and organized in several layers of abstraction. Base classes,

[10]The currently latest version of Orocos is 2.6. With the about-to-be-released version 2.7, the interoperability of ROS and Orocos will be extended.

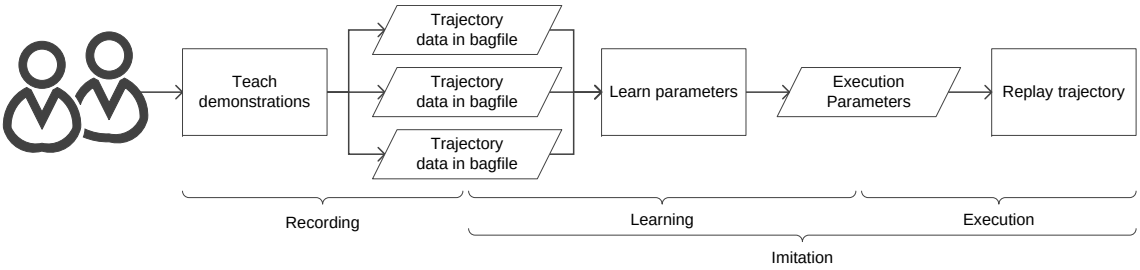

Figure 34: Basic sequence of learning a trajectory. This consists of the recording, learning and execution phase.

templates and interfaces support the generic architecture where various implementations are possible.

4.6.1 Basic framework

To prevent users from tedious, repetitive and error-prone low-level programming, several generic components were developed as a basic framework, encapsulating low-level code and offering higher-level interfaces. In the following, these components are described in more detail. See Figure 35 for a quick overview.

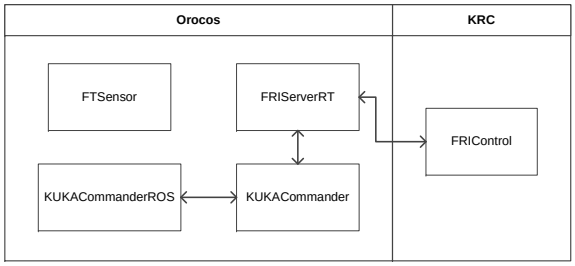

Figure 35: Basic software framework for the LWR.

The Orocos component `FTSensor` handles the communication with the F/T-sensor. It initially sends a start request and subsequently receives all measurements over a UDP Ethernet connection. The measurements are provided to other components using output ports. The status of the sensor is constantly surveilled and errors are logged.

The `FRIServerRT` is also an Orocos component and heavily based on [57]. It is responsible for establishing and maintaining the UDP connection as well as for handshaking with the KRC. All measurements and current parameters from the incoming FRI messages are output at the ports of the component. Similarly, the component reads from its port to fill the appropriate fields of the outgoing FRI message.

Although the communication details are hidden by the `FRIServerRT`, many desired functions still require several steps or are even impossible to achieve with plain FRI. Therefore, the Orocos component `KUKACommander` (on the external computer) and the KRL program `FRIControl` (on the KRC) were designed to further

simplify the work with the LWR, by offering higher-level functionality. First of all, `KUKACommander` offers setter and getter methods for various parameters, such as `getCurrentControlMode` or `setCartesianImpedance`. In order to access other LWR functionalities, FRI itself is not enough and `KUKACommander` has to communicate with `FRIControl` using the aforementioned FRI variables (Section 4.3). This allows the user program to switch between states (e. g. `switchToCommandState`) and modes (e. g. `setControlMode`). Additionally, it is possible to execute linear and point-to-point motions on the KRC (e. g. `CartesianLINMotion`). `KUKACommander` is a new development, `FRIControl` is based on [58] and [59].

It was preferred to do the main development in ROS rather than in Orocos. The reason for this is the greater flexibility and the huge amount of readily available packages for this middleware. Therefore `KUKACommannderROS` provides an interface to all functions of `KUKACommander` to ROS using services[11].

4.6.2 Recording

The recording phase requires only one additional component, the ROS package `Recorder` (see Figure 36). It initially queries the user for the trajectory type (joint or Cartesian) and for the inclusion of force measurements. Thus the ROS topic type of the recorded messages varies for different trajectory types. In Figure 36, the message type is for example “PoseMsg”.

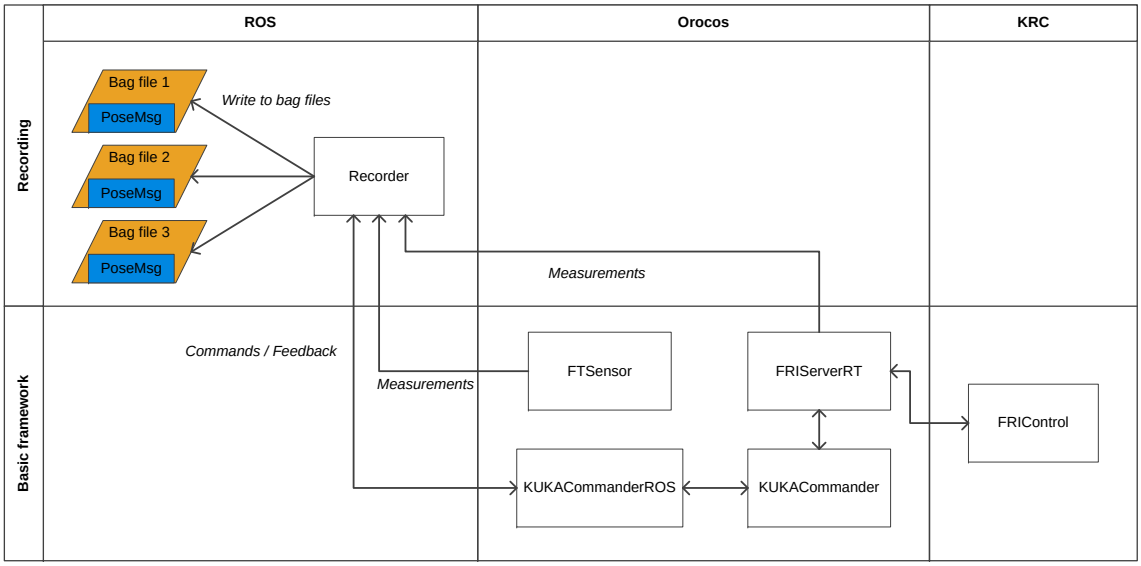

Figure 36: Software components of the recording phase.

For the rest of the procedure, the arm is put in a mode providing gravity compensation. In this mode, the LWR can be moved easily. The joints are actively compliant and compensate gravitational forces. After moving the robot to the start position,

[11] Services are public functions of a ROS node, which can be called with an arbitrary number of parameters and return values. They are typically intended for longer lasting tasks. One disadvantage is that a service call blocks the execution of the caller.

all measured positions (and eventually forces), published from `FRIServerRT` (and `FTSensor`), are recorded to ROS bag files[12]. After one demonstration, the program allows to return to the start position and record further demonstrations in separate bag files.

4.6.3 Imitation

The learning and execution phase are currently bundled together, which means that each time a demonstration is repeated, it must be learned beforehand. Nevertheless, these phases they are separated in several packages and components, as it is shown in Figure 37. This the most sophisticated part of the software and is here referred to as "imitation" (see Figure 34).

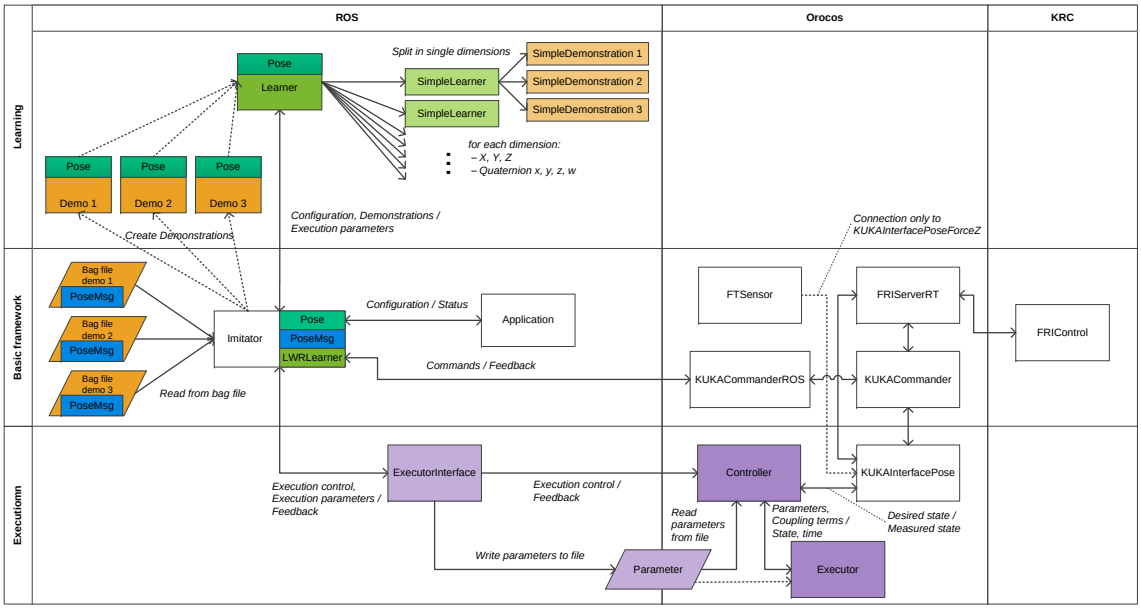

Figure 37: Software components of the imitation phase.

Representation A demonstration is represented by the `Demonstration` class, holding a trajectory (see Figure 38). A trajectory is made of a vector of timestamps and objects of the class `Triple`. A `Triple` is a tuple consisting of a position, velocity and acceleration object, which are derived from the base class `Item`. These items hold joint positions, a pose or pose and force, depending on the user's choice of the derived class. This item type is defined by a C++ template class for `Demonstration` and `Triple`, being `Pose` in Figure 37 and Figure 38. A `Demonstration` can be converted to a vector of `SimpleDemonstration` objects. A `SimpleDemonstration` has four equally sized vectors for time, position, velocity and acceleration, but contains only one dimension of an `Item`. A `Position` item for example consists of the dimensions

[12]Bag files are special files in which ROS can store messages even at high frequencies including their timestamps. ROS can also play back from bag files and offers a library to access the data with C++.

X, Y, and Z is therefore converted into three `SimpleDemonstration` objects. For more details, see Appendix A.

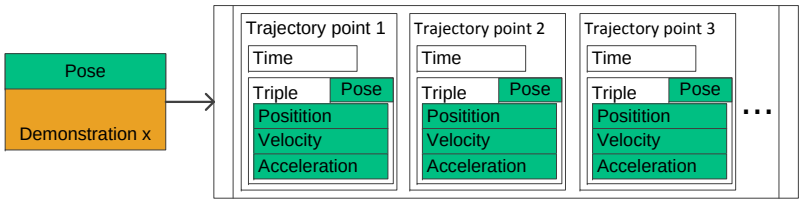

Figure 38: A demonstration contains a trajectory of timestamps and tuples of position, velocity and acceleration.

These classes also hold several methods. A demonstration can be trimmed or calculate velocity and acceleration from only the temporal and positional data. For this, items report about their distance to other items or calculate their temporal derivative. There are also static functions to generate `Item` objects from ROS messages (see Appendix A).

Imitation phase The application creates the `Imitator` and defines all its parameters, such as the prefix of the bag file names, stiffness or number of basis functions. All these values can be configured in a ROS launch file. This allows a change of parameter settings without recompilation. The application also controls the execution flow and informs the user about the current status.

The `Imitator` is the central class in ROS, coordinating all actions. It takes three template classes: the learning class, the item type and the topic type of the bag file (in Figure 37 these are `LWRLearner`, `Pose` and `PoseMsg`). The `Imitator` reads from the bag files and creates a `Demonstration` for each of them. The demonstrations are prepared (i. e. they are trimmed and velocity and acceleration data are calculated, see Section 3.1.2) and passed to the learning class, together with the desired parameters. Having received the execution parameters from the learner, they are forwarded to the `Interface`. The execution is initiated and feedback is forwarded to the application.

Learning phase The only existing learning class so far is the `LWRLearner`. As it can later be seen for the `Executor`, the `LWRLearner` works with vectors of double typed values, representing a state with several dimensions. The reason for that is to allow generalization. For example, the task of the `Item` classes to convert a pose into this kind of vector. Therefore, the learner works with `SimpleDemonstration` objects instead of `Demonstration` objects. The actual learning takes place in the `SimpleLWRLearner`, implementing the equations from Section 3.1.2. Each dimension has its own simple learner (they all share the same parameters) and retrieves the simple demonstrations for its dimension. Thus the task of the `LWRLearner` is to first separate the demonstrations into their dimensions and later put the results of the different simple learners into nested vectors (see Appendix A).

Execution phase Being part of the execution phase, the tasks of the `Interface` are to prepare the execution data for Orocos and to communicate with the `Controller`. It offers the functionalities of the `Controller` to other ROS components in form of so called ROS ActionServers[13]. All execution parameters are stored in an object of the `ExecutionParameters` class. This allows the parameters to be serialized and stored in a file. Only the file name is passed to the `Controller`, which can then retrieve the object again. This serialization procedure will in the future allow learned motions to be stored permanently after learning.

On the Orocos side, the `Controller` plays the central role. The `Controller` retrieves the execution parameters and passes them to the `Executor`. In each cycle, the next desired state is queried from the `Executor` and forwarded to the `KUKAInterface`. The `Controller` also calculates the coupling terms (currently only temporal coupling is used). For this, feedback about the current state is received from the `KUKAInterface`.

The only task of the `Executor` is the calculation of the force term and with that the desired state. It initially retrieves the execution parameters. During the execution, also the coupling terms can be set.

As the `Controller` only calculates the desired state, but does not not know how to interpret this data, another interface is needed. The `KUKAInterface` is the counterpart to `Item` on the execution side. `KUKAInterface` serves as abstract base class for implementations such as `KUKAInterfaceJoints` or `KUKAInterfacePose`. The interfaces convert the retrieved state vector into objects like `JointState` or `Pose`, which are directly send to the `FRIServerRT`. Similarly, the current measurements (coming from `FRIServerRT` and `FTSensor`) are serialized to vectors and returned as feedback to the `Executor`. Next to that, the interfaces are also responsible for switching to the correct command mode (e. g. Cartesian stiffness controller) and set the correct impedance values using the `KUKACommander`.

Helper functions Not shown in the diagrams are the several helper classes. `Canonical` is an abstract base class for the canonical system. Implementations are `Exponential` and `PhaseOscillator` for, respectively, discrete and cyclic motions. Similarly, `Distribution` is an abstract base class for the basis functions. `Gaussian` is here the implementation for discrete motions and `vonMises` for cyclic ones. The objects of these classes can be serialized, as they are part of the execution parameters.

4.6.4 Implementation details

Although the description of all details of the implementation would go far beyond the scope of this thesis, some details are noteworthy. Especially the handling of rotations with DMP requires some care, but also the observation of the execution.

It was mentioned that the `Controller` receives feedback from the `KUKAInterface`. This requires some more explanation. The `Controller` sends the desired state in

[13] ActionServer/ActionClient as part of ROS actionlib are similar to services. The main difference is that a call of an action does not block the execution of the caller. In addition, ActionServers can give feedback about the current status and the ActionClients can preempt a task.

form of a vector of type double, which the different interfaces interpret for example as position, pose, or joint state. The `Controller` requires a state feedback, that is the current (measured) state, also in form of a vector of type double. This feedback is especially useful for the perturbation handler (see Section 3.2.2). Therefore the interfaces must convert the current state (in form of position, pose, etc.) back into this type of a vector. However, the interface is free to return not the true measurements, but to modify them, as will be shown in the next paragraphs.

A common representation for rotations are unit quaternions [60, p. 54]. Different rotational representations, such as Euler angles have the disadvantage to "jump", not to be unique (due to facts such as $0° \triangleq 360°$) and to have singularities. This introduces problems for the function approximator, which cannot handle these discontinuities appropriately. Other representations, such as rotation matrices are computational heavy and memory consumptive. Quaternions, consisting of four elements, do not have these kind of drawbacks. That is why quaternions have been chosen in this work to represent rotations.

However, following such representations, two issues must be addressed. First, two quaternions result in the same orientation, when $q_1 = -q_2$. FRI gives the orientation as rotation matrix, which is converted to a quaternion. In some cases, this conversion produces a series of quaternions, whose sign can suddenly change. This also causes undesired discontinuities. Therefore, when creating a `Demonstration` from a bag file, a function iterates over all quaternions and switches the signs where needed.

The second issue concerns the feedback state of quaternions. From pairwise comparison of the four elements of the desired quaternion and the measured quaternion, the generic `Controller` cannot correctly determine the relative error of the angle. Therefore, the interfaces `KUKAInterfacePose` and `KUKAInterfacePoseForceZ` modify the quaternion which is returned as feedback. They take the desired and measured quaternion and calculate the relative quaternion between the two (i. e. the relative orientation). From this relative quaternion, the rotation angle is determined, which corresponds to the absolute error of the angle. Three elements of the quaternion feedback state are set to the desired value, thus not producing a relative error. The forth component is set to a value causing the appropriate relative error.

A final note shall be given on the calculation of the weighting factor w_i. The values in the enumerator of Equation (25) can get smaller than the numerical accuracy of double typed variables. This results in C++ in a value `NaN` (Not a Number). This is detected and the corresponding weighting factor w_i is set to zero.

4.7 Discussion

Evaluating a software architecture is not easy. In this thesis, no specific experiment was performed to quantitatively support design decisions. However, the framework was not developed for high-performance nor to be immune against external attacks. The focus of the design was flexibility, modularity and abstraction.

Therefore the software not only uses established approaches of *Object-oriented programming* (OOP), but also relatively new concepts such as template classes. After the basic architecture had been written (in code), the further development could be

performed quite fast. For example using the abstract base class `KUKAInterface`, the coding of the implementing child classes such as `KUKAInterfaceJoints` or `KUKAInterfacePosition` was only a matter of a few hours.

This was confirmed by the implementation of the basic DMP algorithms. The extension for in-contact tasks could be implemented with hardly any change in the software architecture. The design is flexible enough to support a variety of implementation details. For example, modularity allows to test different kind of canonical systems by only implementing a new child class inheriting from the abstract canonical base class.

Nevertheless, one part still needing significant improvements is the interface between Orocos and ROS. This interface partially uses ROS topics and partially uses ROS services, which are forwarded to other classes using services. This implementation is bounded by the limited compatibility between ROS and Orocos. With the new version of Orocos (which will probably be released soon), this compatibility is heavily increased. This release should be used to improve the disorganized current interface and to upgrade the ROS version to Hydro. Instead of using several ports, which translate to topics in ROS, ActionServer/ActionClients could be used for asynchronous tasks giving feedback about the current status. The communication should also be transparently passed through the different layers of abstraction, instead of establishing pairwise encapsulated communication between the layers.

Another issue is the current limitation of the imitation phase. The learning and execution phase are tied together, so all movements need to be newly learned each time they are executed. The solution for that does not require dramatic extensions in the code, as the functionality to store the learned parameters is already existing.

5 Experiments

The DMP framework has demonstrated promising results in other implementations, as was shown in Section 2.4.1. Nonetheless, the implementation, together with novel perturbation and force handling strategies, has required significant technical work and careful evaluation of the results. Therefore, some experiments were laid out, in order to validate our experimental platform.

The target of the experiments is to verify the DMP framework with the developed extensions. First the general functionality is tested, also with several demonstrations. Then the perturbation handler as well as the force and velocity controller are evaluated. The main research questions are, whether it is safe and unproblematic to interfere the execution and whether the system is able to at least qualitatively reproduce not just a trajectory, but also a demonstrated force profile.

The results of the experiments mainly focus on the trajectory (poses) and force data, which was recorded using ROS bag files. The measured poses are the position and orientation of the tip of the F/T-sensor (not the tip of the pen), relative to the base of the LWR. The reason for this is to make the measurements from the F/T-sensor comparable to those from the LWR. There are five different examined data sets, which have to be distinguished:

1. **The demonstrated data:** This is the data recorded from the user and the therefore the input for the LWR learner. The data has already been prepared. Preparation is especially done for the quaternions of the orientation, as is described in Section 4.6.4.

2. **The theoretical data:** This is the output of the transformation system, thus with neither hardware nor controller involved and the coupling terms set to zero.

3. **The calculated data:** This is also the output of the transformation, but with respect to the dynamic coupling terms. This means that the calculated trajectory takes into consideration the effects of the perturbation handler during the execution.

4. **The experimental data:** This is the data recorded during the execution phase. It is the measured pose of the arm or the measured force of the F/T-sensor, not the output of the controller.

5. **The commanded data:** This data-set only exists for forces. These are the actual commands sent to the KRC. For the trajectory, the calculated and commanded values are identical (and therefore only the calculated data is shown here). However for forces, two additional controllers (force and velocity controller, see Section 3.2.3) modify the calculated values before commanding them to the KRC.

For quantitative comparisons, four metrics from [27] are used. M_1 is the RMS error between two trajectories. Similarly, M_2 is also the RMS error, but with the

trajectories being temporally aligned using the DTW approach [26] (see Section 2.3.2). The M_3 metric is the norm of jerk of a single trajectory. The time t_{learn} to learn the model parameters is the metric M_4, calculated on a processor with 3300 MHz. For the metrics M_1, M_2 and M_3, only the position data is used, not the rotation or force data.

5.1 Imitation of a circular motion

The goal of this experiment was to validate the results achieved with the software and hardware infrastructure and test the basic DMP implementation. Next to the comparison of the demonstrated and learned trajectory, also the learning time is of interest.

5.1.1 Experiment

In the recording phase of this and all other experiments, the demonstration was taught as described in Section 4.6.2. The recording took always place on the Cartesian level. The poses were sampled with a frequency of 200 Hz (5 ms cycle). The arm was grabbed and moved in a horizontally aligned circle. Only one demonstration with $M_1' = 1851$ elements and $N_D = 7$ (3 components for positions and 4 for quaternions) was shown. Figure 39 shows the setup of the experiment.

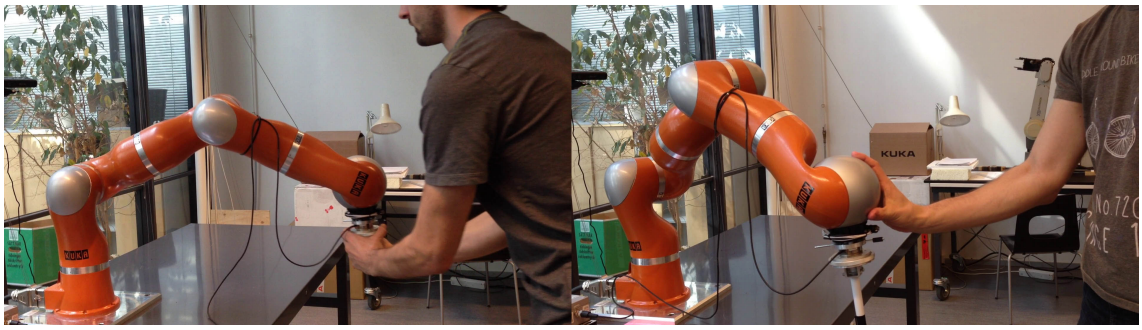

Figure 39: Setup of the basic experiments. In the left image, the trajectory is recorded. The right image shows an example perturbation (the motion is stopped) during the execution phase.

For the imitation phase, the following parameters were used. The stiffness of the Cartesian impedance controller (k_c in Equation (29)) was set to 2000 $\frac{\text{N}}{\text{m}}$ for linear and to 300 $\frac{\text{Nm}}{\text{rad}}$ for angular movements. The damping values d_c were 0.7 $\frac{\text{Ns}}{\text{m}}$ and 0.7 $\frac{\text{Nms}}{\text{rad}}$ for, respectively, linear and angular movements. The parameter α_z of Equation (2) was set to 2000 and β_z was chosen for critical damping ($\beta_z = \alpha_z/4$). The number of basis functions was $N_w = 50$ for all dimensions. The exponential decay of the canonical system (Equation (7)) started at $x_0 = 1$ with a decay parameter of $\alpha_x = 3$. The LWR was commanded with a FRI communication rate of 500 Hz.

5.1.2 Results

The arm repeated the demonstrated motion successfully after a learning time of $t_{\text{learn}} = 0.0403\,\text{s}$. Figure 40 shows the different trajectories in two 3 D diagrams. The left plot only displays the positional data, while the arrows in the right plot also give the tool direction, i. e. the orientation. These arrows are equally spaced in time with a time step of $\Delta t = 0.2\,\text{s}$. In Figure 41, all seven components of the poses are plotted against time.

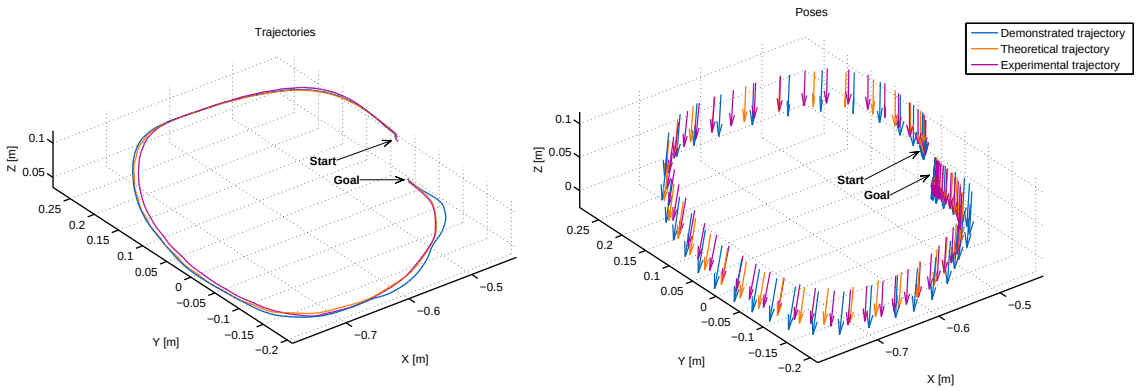

Figure 40: Execution of a demonstrated circular motion: 3 D view on the trajectories.

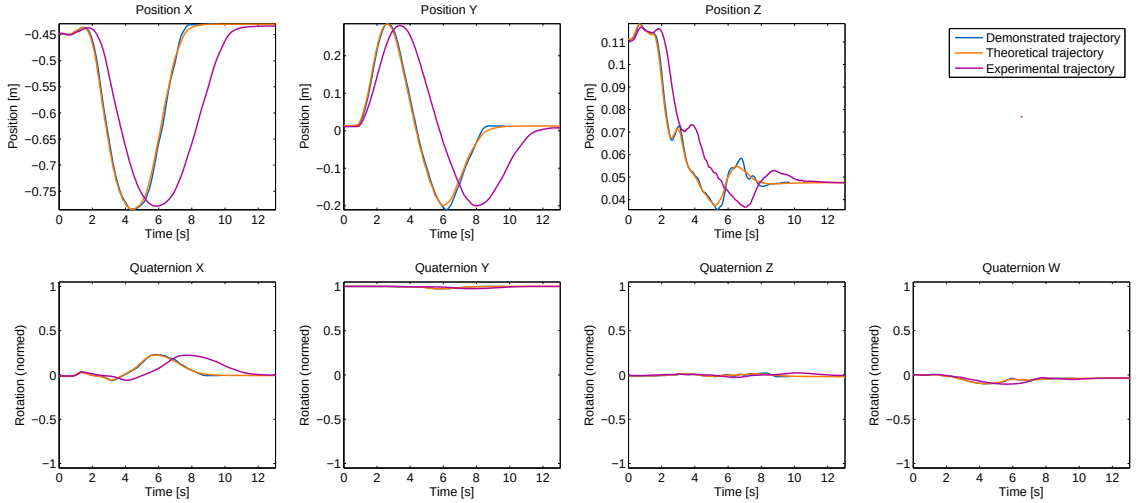

Figure 41: Execution of a demonstrated circular motion: All components of the poses are shown over time.

The first qualitative result from Figure 40 is that all three trajectories are close together. Only at the end of the trajectory, some details of the demonstration were lost. This is due to the fading of the force term (see Section 3.1.1), leaving only the attractive force of the goal.

Another important fact arising especially from Figure 41 is that the theoretical trajectory closely mimics the demonstrated one. Only in case of rapid changes, the

theoretical trajectory smooths out the demonstration. This can for example be seen in the Z dimension between $t = 5\,\text{s}$ and $9\,\text{s}$.

Obviously, the experimental trajectory deviates from the demonstrated and theoretical trajectories when considering the temporal dimension as shown in Figure 41. The difference increases continuously towards the end of the execution. The cause of this is the perturbation handler described in Section 3.2.2. As the arm always remains slightly behind the desired pose, the canonical system is slowed down, until the arm can follow the desired trajectory. This offset with respect to the desired pose may be explained by considering the Cartesian impedance controller, which is set to be critically damped. Therefore, the set point is only reached asymptotically. The calculated trajectory has not been recorded for this experiment, that is why the evaluation of experimental trajectory is further developed for the experiments in Section 5.4 and 5.5.

Table 3 gives the quantitative results for this experiment for the metrics M_1, M_2 and M_3. The M_1 data confirms the big offset between the experimental trajectory and the demonstrated/theoretical one. However, the RMS errors between the temporally aligned trajectories are all low ($< 6\,\text{mm}$). It can also be seen that the experimental trajectory cannot as closely mimic the demonstrated one as the theoretical trajectory does. The M_3 metric gives some more interesting results. The jerk of the demonstration is reduced a lot for the theoretical trajectory. However for the experimental trajectory, the jerk is much higher than for the demonstration. One explanation might the the different rate of communication during the recording and the execution phase. The timestamps used for the evaluation are those generated by ROS, which is not real-time capable. Therefore there might be a jitter in these timestamps, causing the high jerk.

Comparison	$\mathbf{M_1}$ **[m]**	$\mathbf{M_2}$ **[m]**	**Trajectory**	$\mathbf{M_3}$ **[m/s^3]**
Demonstration/Theory	0.0105	0.00247	Demonstration	283.3
Demonstration/Experiment	0.1662	0.00598	Theory	15.43
Theory/Experiment	0.1700	0.00396	Experiment	988.7

Table 3: Quantitative results for the first experiment

5.2 Multiple demonstrations

This experiment focuses on the handling of multiple demonstrated instances of the same desired behavior. The system should be able to generalize over the different demonstrations. Therefore, the resulting trajectory should be qualitatively similar to all demonstrations.

5.2.1 Experiment

The experiment was conducted following the same protocol as the previous one. It made use of the same parameters and of the previously recorded trajectory. The difference was the second demonstration with $M_2' = 1909$ elements, which was recorded

and provided to the learner. The two demonstrations were similar, but the size and duration varied a bit.

5.2.2 Results

During the execution, the arm performs a circular movement qualitatively similar to the demonstrations. Figure 42 and 43 report the plots for this new experiment, qualitatively similar to the previous one. Only the orientation data is omitted for brevity. The two demonstrations differ in duration ($\tau^1 = 9.26\,\mathrm{s}$ and $\tau^2 = 9.54\,\mathrm{s}$) and spatial extension (e. g. for the Y dimension $A^1_{\max} = 0.496\,\mathrm{m}$ and $A^2_{\max} = 0.531\,\mathrm{m}$), but also in shape. The learning time was $t_{\mathrm{learn}} = 0.0842\,\mathrm{s}$.

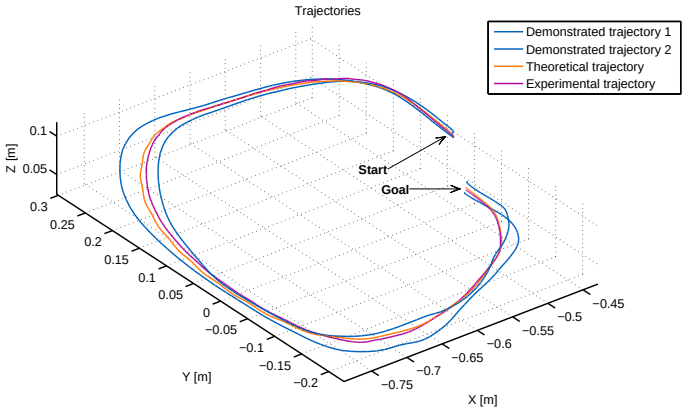

Figure 42: Learning from two demonstrations: 3 D view on the trajectories.

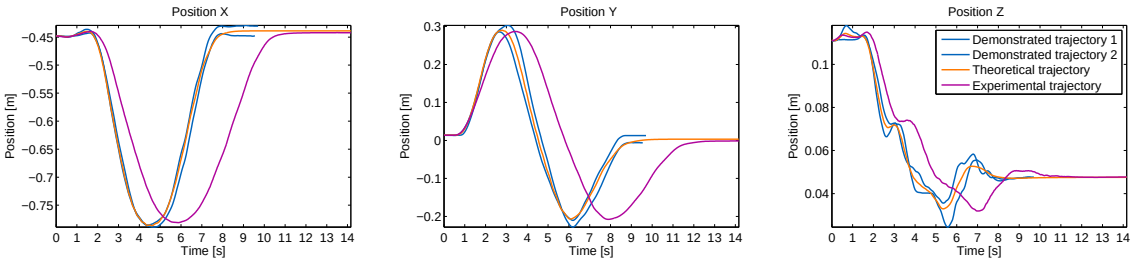

Figure 43: Learning from two demonstrations: All positional components are shown over time.

First of all, this experiment confirms the results from the previous experiment. All trajectories are relatively close together, while the execution speed is slower than the speed during the demonstration.

When comparing the 3 D data in Figure 42 and the trajectory components over time in Figure 43, the effect of a second demonstration can be seen. The learned trajectory keeps the overall shape of the two demonstrations, while being much smoother (the effect is particularly visible for the Z component). Thus, the learning process successfully takes the data from all demonstrations into account. The spatial

scale of the experimental trajectory is about the average between the two demonstrations. This results from the fact that during the execution, the scaling term $\xi(x)$ is set to the average of all scaling terms of the demonstrations. So, although no experiment was conducted which used varying values for the scaling term, it seems that trajectories can successfully be scaled using this term.

5.3 Perturbation handling

The goal of this experiment is to verify the appropriate handling of external disturbances that physically disturb the movement of the robot during execution. On the one hand, the question is whether it is safe for a user to interfere the execution. On the other hand, the effect on the trajectory is of interest, for example whether some details are left out.

5.3.1 Experiment

The input data as well as the parameters for this experiment were identical to the first experiment (execution of a circular motion, Section 5.1). Yet, the robot was stopped manually two times during the execution phase. A human experimenter grabbed the tip of the arm at these situation and forced the movement to stop (see Figure 39, right). After a short period of time, the robot was released again.

5.3.2 Results

The robot acted again as expected again accordingly to DMP theory [33]. The positional data resulting from this experiment are shown in Figure 44 and 45, according to the previous experiments. The arm was being hold between $t = 3.7\,\text{s}$ and $5.2\,\text{s}$ and between $t = 9.5\,\text{s}$ and $10.6\,\text{s}$.

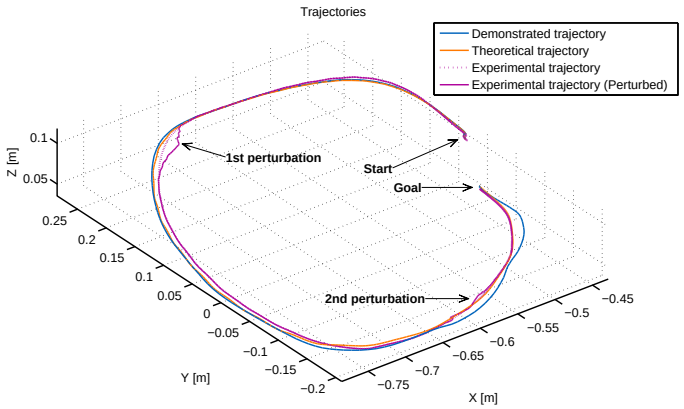

Figure 44: Perturbation of a circular motion: 3 D view on the spatial trajectory.

By looking at the first plot of Figure 45, it can be clearly seen that the execution comes to a stand at the mentioned time intervals. After the two perturbations, the arm continues its movement without any sudden jump. The trajectory is resumed

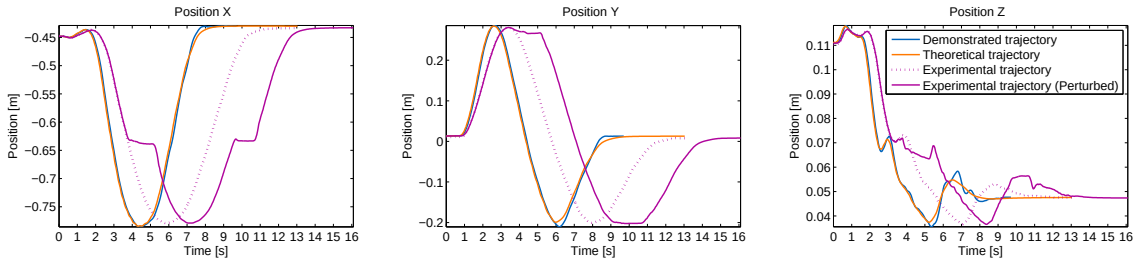

Figure 45: Perturbation of a circular motion: All positional components are shown over time.

from where it has been stopped, so no detail of the trajectory is omitted. Even more: the arm smoothly compensates for its displacement, as can be seen for the first perturbation in Figure 44. This is the effect of the developed perturbation handler, described in Section 3.2.2.

The force needed to interrupt the execution was not measured. However, the (subjective) sensed force necessary to stop the robot and keep it still with the hand was very low. Also, due to the low speed, no hard impact was perceived.

5.4 Writing on a notepad

The purpose of this experiment is the evaluation of the in-contact task algorithm, especially the force-controller. The system shall show its capabilities to follow both the demonstrated force profile and the trajectory. It should smoothly switch between motions in-contact and not in-contact.

5.4.1 Experiment

For this experiment, the cap of the tool pen (a whiteboard marker) was removed and a notepad positioned on a sturdy box elevated to about 13 cm above the table. Holding the robots arm from above the F/T-sensor, the robots name “KIM” was drawn on the pad during the kinesthetic teaching phase. Particular attention was paid to pressing the pen down with minimal necessary force. The number of elements of the demonstration were $M_1' = 29845$ and the dimensionality $N_D = 8$ (seven for the pose and one for the force in Z direction). The setup of the experiment is shown in the beginning of the thesis in Figure 1.

Most of the parameters were set to the same values as for the previous experiments. However, the number of basis functions was increased to $N_w = 250$, to cope for the longer duration. For the sake of accuracy (especially at the end of the trajectory), the decay parameter was lowered to $\alpha_x = 1.1$. During both the recording and execution phase, the FRI communication rate was 500 Hz. In addition to the pose data, also the force in Z direction of the F/T-sensors was recorded. The interface utilized was `KUKAInterfacePoseForceZ`, which makes use of the algorithms described in Section 3.2.3. This interface was modified, to limit the desired force f_{DMP} to 3 N. The intention was to limit during the execution the effect of erroneous measurements or accidentally exerted high forces experienced during the demonstration.

5.4.2 Results

In this experiment, the system has to model an additional force component and a much more complex trajectory. Therefore, the learning time increased to $t_{\text{learn}} = 3.08\,\text{s}$. The motion in 3 D Cartesian space is shown in Figure 46 (left). The plot to the right shows the orientation of the pen. This orientation only slightly varies around the negative Z-axis (in the base frame). Therefore, the corresponding plots are omitted in Figure 47 and in the rest of the analysis, focusing only on the positional components over time. An overview of the force profiles is given in Figure 48. Figure 49 zooms in to offer a detailed view during the execution of the 'K' character. The force data from the execution is plotted in Figure 50.

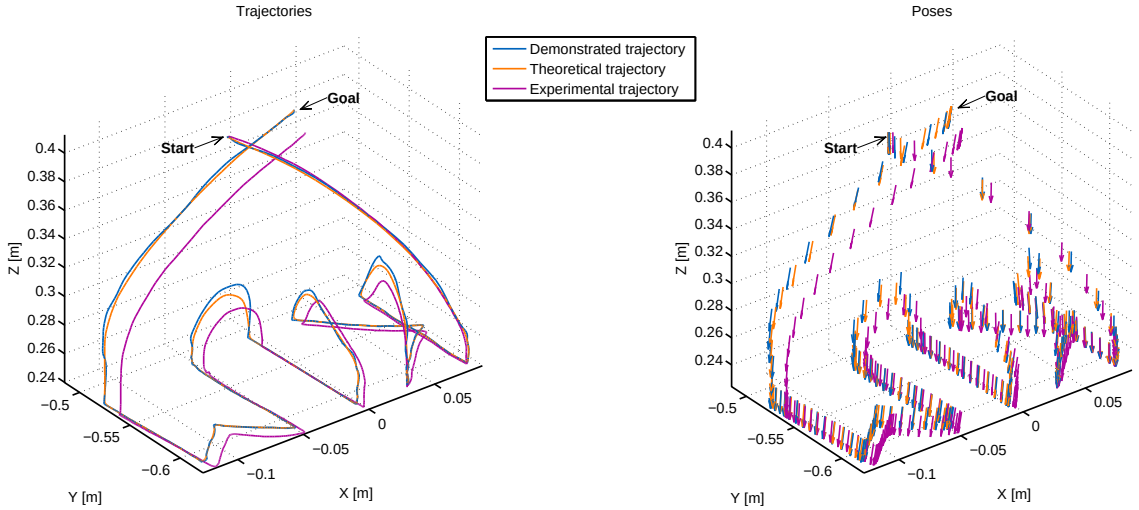

Figure 46: Writing "KIM" on a notepad: 3 D view on the trajectory.

The number of basis functions has been significantly increased compared to the previous experiments. Nevertheless, due to the length of the demonstration, the temporal density of the basis functions $d_{N_w,\tau} = N_w/\tau$ is even smaller: $d_{N_w,\tau} = 4.19\,\text{s}^{-1}$ compared to $d_{N_w,\tau} = 5.40\,\text{s}^{-1}$ in the previous experiment. Thus the expected accuracy is a bit lower. However, this density is only a theoretical average, as the basis functions are equally spread in the phase variable and not in time (see Figure 26).

Pose trajectory The result is again a theoretical trajectory that closely mimics the demonstration, even towards the end of it. In this similarity at the end, the effect of a low value for α_x can be seen, which causes the force term to keep its influence during the whole execution. Only in the case of rapid changes (for example the high peaks in the Y and Z component of Figure 47) the theoretical trajectory is smoothed.

The experimental trajectory offers two insights. First (as in the previous experiments), the execution is slower than desired. However, the difference in execution speed is lower than for the previous experiments. This results from a de-facto deactivated perturbation handler while being in contact. In this case, all feedback values are identical to the desired ones (see Section 3.2.3). Second, especially the Y and Z

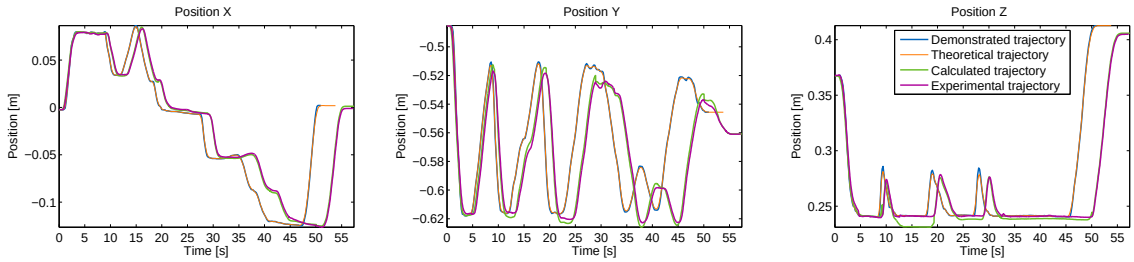

Figure 47: Writing "KIM" on a notepad: All positional components are shown over time.

components of the experimental and calculated trajectory show some deviation from the theoretical one. This offset increases every time the contact is lost. This can also be explained: Since the correctness of the position is not controlled while in contact (as just explained), the executed trajectory can deviate from the desired one. The friction between the pen and the surface contributes to this deviation, as it exerts an opposing force to the attractive force of the set point of the Cartesian impedance controller. When the contact is lost, this offset is stored (see again Section 3.2.3). However, the error introduced by this effect is at most 0.75 cm. Nevertheless, the offset of the Z component cannot be interpreted in the same way. In this case, the reason can be found by taking a closer look at what happens with the Z component as the contact is lost the first time ($t = 9.2\,\text{s}$, Figure 47). The synchronized theoretical trajectory rises up to $z = 0.25$ and right after that jumps back to $z = 0.24$. Thus, the storage of the positional offset, which should be done when the contact is lost, takes place too late.

The quantitative results of the metrics M_1, M_2 and M_3 are shown in Table 4. The M_1 data confirm the qualitative results. The demonstrated and theoretical trajectory are very similar. The experimental trajectory deviates further from the other two, but not as much as for the first experiment (Section 5.1.2). When temporally aligned, the RMS error between the demonstration and theory is lower than 1 mm (M_2 metric). However, the experiment is not able to reach that accuracy. On the one hand, the jerk of the theoretical trajectory is by three orders of magnitude lower than the demonstrated trajectory (M_3 metric). On the other hand, the jerk of the experimental trajectory is about as high as for the demonstrated trajectory.

Comparison	**M_1 [m]**	**M_2 [m]**	**Trajectory**	**M_3 [m/s^3]**
Demonstration/Theory	0.0013	0.00073	Demonstration	394.5
Demonstration/Experiment	0.0388	0.00626	Theory	0.452
Theory/Experiment	0.0383	0.00629	Experiment	419.9

Table 4: Quantitative results for the writing experiment

Force profile Section 3.2.3 has introduced the force controller, which is active while in contact. The output of this controller is shown in Figure 48 and 49 with the label "Commanded forces". These are the force commands, forwarded to the Cartesian

impedance controller (Z component of f_{des} in Equation (29)). The limitation imposed on f_{DMP} (see Section 5.4.1) can be seen in the lower threshold of the theoretical forces at $-3\,\text{N}$. The force data during the execution phase is given Figure 50.

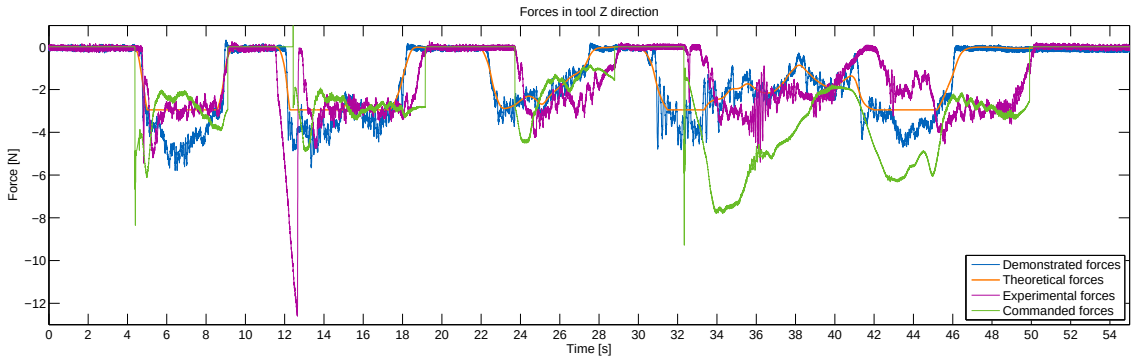

Figure 48: Writing "KIM" on a notepad: Force profiles over time.

Looking at the demonstrated forces in Figure 48, the measurements show oscillations with an amplitude of around $\pm 0.5\,\text{N}$ and a frequency of about $8\,\text{Hz}$, superimposed by weaker, higher frequency oscillations. The oscillations with lower frequency only occur while the pen is in contact. The experimental forces show the very same characteristics. This might be due to the limitations of the F/T-sensor in use, or might also be an effect of the dynamics of the (frictional) contact forces between pen and pater. While the experimenter tried to press the pen down with constant force, it can be seen that this is a task difficult (or even impossible) to achieve.

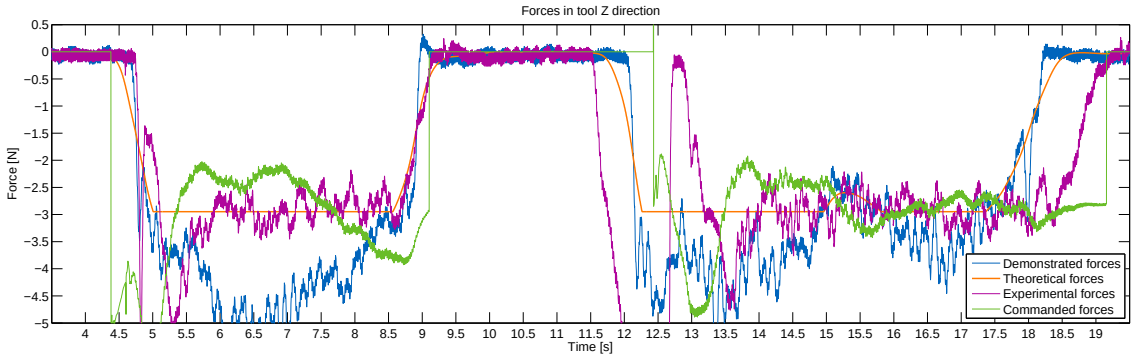

Figure 49: Writing "KIM" on a notepad: Detailed force profiles for the drawing of the character 'K'over time.

The theoretical forces are much smoother (see Figure 48). The function approximator acts like a low pass filter with a very low cut-off frequency. This is very visible between $t = 34\,\text{s}$ and $41\,\text{s}$ (an interval during which the force limitation did not operate). While this filtering is most of the time advantageous (as the high frequent oscillations are not desired), it can be problematic at the touch down and release points. In fact, the rise and fall time in this experiment was at least $t = 0.5\,\text{s}$ for a force difference of $3\,\text{N}$. If the arm is taught to move immediately after contact loss, this movement could take place while still in contact in the execution phase.

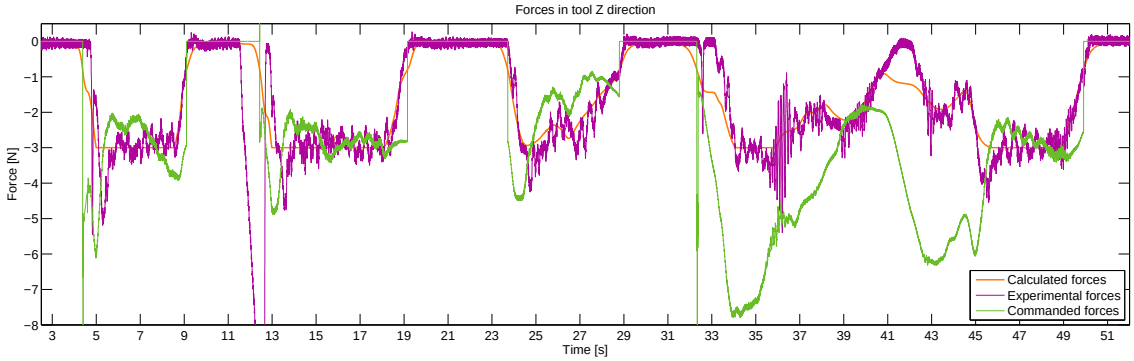

Figure 50: Writing "KIM" on a notepad: Detailed force profiles of the execution phase over time.

Taken into consideration the fluctuations of the F/T-sensor measurements, the experimental forces are close to the calculated ones (see Figure 50). They do also oscillate, but with a similar amplitude and frequency as the demonstrated forces. The quantitative result from the M_2 metric confirms the good visual impression. The RMS error of the temporally aligned demonstrated and experimental force profile is only 0.20 N. Overshoots can mainly be seen at the point the pen gets in contact (e. g. at $t = 5\,\mathrm{s}$). However, there were also comparatively very high force exertion, for example between $t = 11.8\,\mathrm{s}$ and 12.7 s. When looking at the theoretical and commanded forces at this period of time, there is neither a force desired nor commanded. Therefore this was most probably the result of getting in contact too early, i. e. before a force exertion is desired. This was likely caused by the wrong offset stored for the Z component, as has been described in the previous subsection.

The imperfections of the KUKA force controller and the performance of the ID controller, introduced to compensate for them (Section 3.2.3) are visible in Figure 50. While the experimental forces (the actual exerted forces) match more or less the theoretical ones, the commanded forces deviate quite far from them. That means that a force with a large varying offset from the desired force has to be commanded in order to exert a force that fits the desired one.

5.5 Writing on a lowered notepad

This experiment tests the ability of the system to generalize from the previous one (Section 5.4). In particular, its aim is to evaluate the capabilities of the designed velocity controller (see Section 3.2.3) and the general capability of shifting the overall trajectory. Next to this, also the compensating effect of the force controller to the erroneous dynamic force term is of interest.

5.5.1 Experiment

This experiment used the same input data and parameters as the previous experiment. The difference appears during the execution phase, when the box, which elevated

the notepad during the learning phase, was removed. Therefore the height of the contact point is about 13 cm lower than expected (see Figure 1).

5.5.2 Results

The system copes with the height difference without any problems. Figure 51 shows the 3 D data, while Figure 52 plots the positional components over time (including the calculated values). The demonstrated and theoretical forces are given in Figure 53. The force data from the execution phase is plotted in Figure 54. Finally, pictures of the demonstrated and the two experimentally written "KIM" letters are presented in Figure 55.

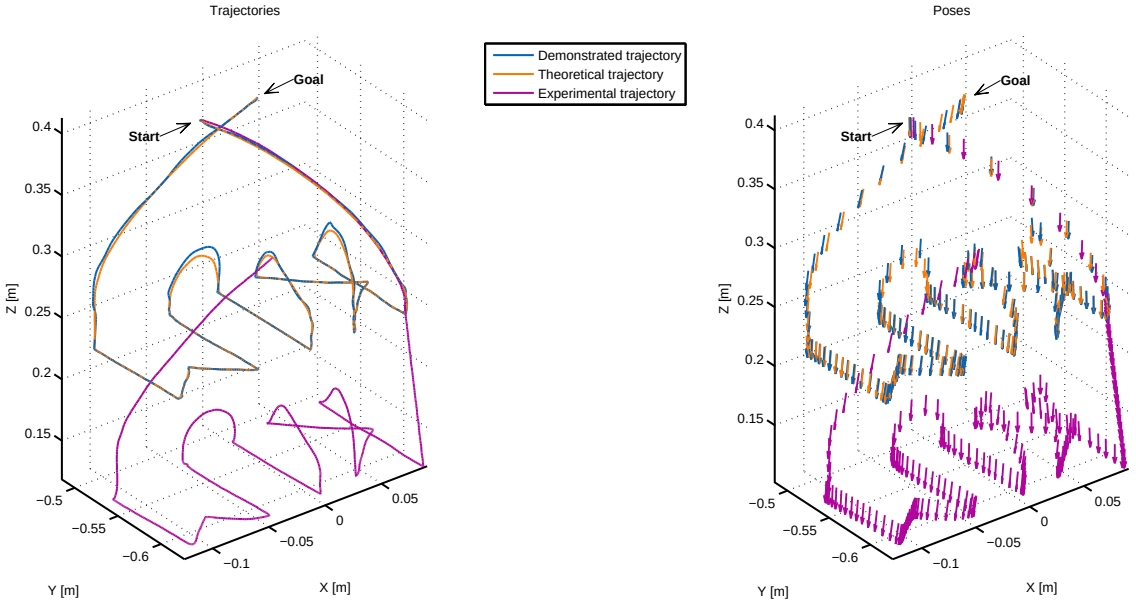

Figure 51: Writing "KIM" on a lowered notepad: 3 D view on the trajectory.

Pose trajectory The 3 D view (Figure 51) already gives a qualitative impression of the success of this experiment. As the notepad had been lowered, the arm moved downwards to the pad's new height. The remaining trajectory is executed relative to this new contact point.

The plots of Figure 52 confirm this result. The Z components moves in a straight line from $t_1 = 4.62\,\mathrm{s}$ and $z_1 = 0.245\,\mathrm{m}$ to $t_2 = 10.98\,\mathrm{s}$ and $z_2 = 0.12\,\mathrm{m}$. This yields an average speed of $v_{\mathrm{avg}} = 1.97\,\mathrm{cm/s}$, which is very close to the desired $v_{\mathrm{approach}} = 2.0\,\mathrm{cm/s}$. The direction of the approach is in tool Z direction. As the tool is not perfectly perpendicular to the plane of the notepad, the approach vector is slightly tilted compared to the normal of the plane. This results in a touch down point which is slightly away from the horizontal projection of the desired one. However, this offset vanishes as the execution proceeds. The shape of the trajectory after contact is similar to the one for a non displaced object. Nonetheless, the problem with the wrongly stored offset is the same as for the previous experiment.

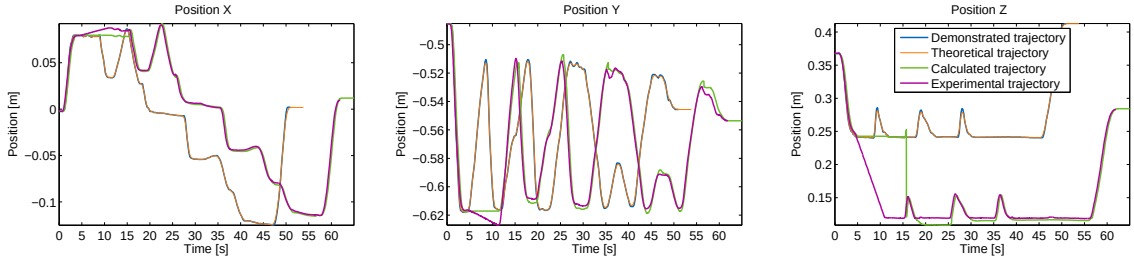

Figure 52: Writing "KIM" on a lowered notepad: All components of the poses are shown over time.

Force profile Also the general characteristics of the theoretical force profile, shown in Figure 53, are the same as for the previous experiment. On the one hand, all high frequent fluctuations are filtered out. On the other hand, the rise and fall time are comparably high.

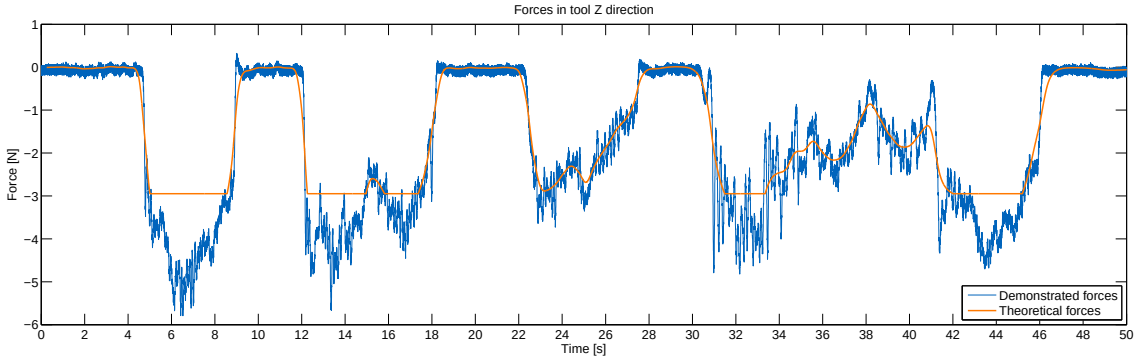

Figure 53: Writing "KIM" on a lowered notepad: Force profiles over time.

Nevertheless, the calculated force profile in Figure 54 shows a new feature. Between $t = 3.2$ and 9.7 s, the development of this force slows down, stops (remains constant) and continues again. This is the effect of the temporal coupling term, which stops the canonical system while approaching the object. The coupling term is here set during the approach phase, intentionally bringing the canonical system to a halt.

The constant offset between the experimental and the calculated force at the beginning of the trajectory is no error. During this approach phase, there cannot be an external force. With a few exceptions, the experimental force profile (plotted in Figure 54) closely resembles the desired one. This is again confirmed by the quantitative result of the M_2 metric, yielding a RMS error of 0.24 N. However, two main issues exist. The wrong offset again results in two undesired force peaks at $t = 17.3$ s and 28.5 s. Another problem is the emergence of strong oscillations during the time between $t = 37.9$ s and 41.9 s. The reason for these oscillations is currently not clear. Interestingly, the demonstrated trajectory also shows increased fluctuation for the same portion of the trajectory.

The commanded force profile in Figure 54 offers further insight on the two controllers presented in Section 3.2.3. During the approach phase, the commanded

force oscillates around $-4.5\,\mathrm{N}$. This is the force causing the arm to move downwards with the desired speed. In Section 3.2.3, the KUKA force controller was said to have mainly pose dependent systematic error. During three time intervals of this experiment (between $t = 10.1\,\mathrm{s}$ and $13.4\,\mathrm{s}$, $18.2\,\mathrm{s}$ and 23.1, $50.8\,\mathrm{s}$ and $53.9\,\mathrm{s}$), the desired force remains constant ($-3\,\mathrm{N}$) and the movement of the pen is along the negative Y axes. When looking at the experimental force during these periods, the measured force is around the desired $-3\,\mathrm{N}$. Yet, the commanded forces linearly decrease each time by about $4\,\mathrm{N}$. This both confirms the systematic error of the dynamic force term f_{dyn} (Equation (29)) and the compensating effect of the developed ID controller. If there were no measurement error, the force measurements would remain more or less constant during these periods, as the exerted force is roughly constant.

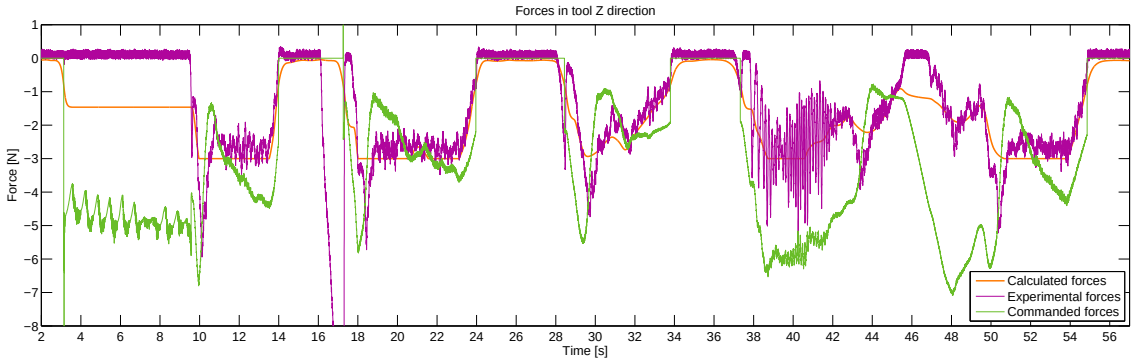

Figure 54: Writing "KIM" on a lowered notepad: Detailed force profiles over time.

A last qualitative evaluation is given by the images of Figure 55. The left image shows the result of the manually demonstrated writing task. The written letters in the middle and right image are created by the robot arm for the notepad in the original and the lowered height, respectively. The similarities of the three "drawings" are obvious. However, one can also see the alteration of the components in the Y direction by comparing the vertical alignment. Even more evident is the different starting point of the letter 'K' in the last image.

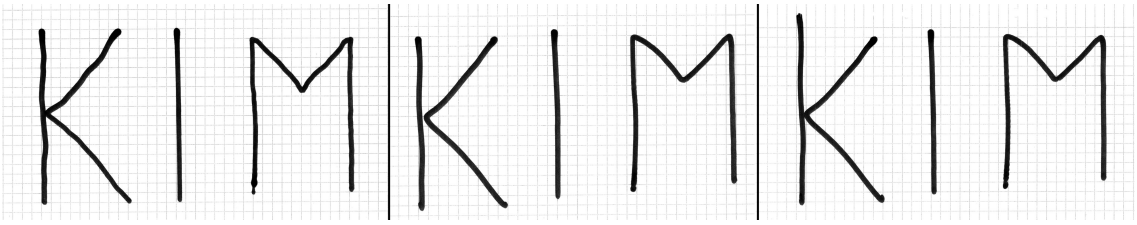

Figure 55: Writing results from kinesthetic demonstration: demonstration (left), reproduction (center) and reproduction on a lowered notepad (right).

5.6 Discussion

In contrast to the generality required for the software architecture, all experiments aim to the evaluation of the DMP framework. Despite the fact that not all features have been investigated, the performance of the major functionality was confirmed.

The first two experiments (Section 5.1 and 5.2) verify the general capabilities of DMP. A simple trajectory can be learned from one or more demonstrations. In the case of multiple demonstrations, the learned trajectory is basically the average over all of them, removing for example undesired jitter.

The third experiment (Section 5.3) proves the successful handling of external mechanical perturbations. When the arm hits an obstacle, the algorithm does not simply stop the motion, but smoothly restores its motion as soon as the perturbation is removed. Furthermore, this mechanism also proves safe for human interactions. Only little force is needed to cause the robot to stop. This reduces the power of an impact and with this the risks of injuries to a minimum.

The writing experiments in Section 5.4 and 5.5 demonstrated the success of the novel in-contact task algorithms. The system is able to learn and exert desired forces. The algorithm handles demonstrations of alternating motions with and without contact. Despite that the dynamic force term shows a varying, pose dependent systematic error, the higher-level controller is able to follow the desired force profile quite accurately. In addition to that, the algorithm smoothly copes with displaced targets. That means, when the desired object of contact is moved along the Z axis of the tool compared to the position in the teaching phase, the arm moves in Z direction until it finds contact. When the new position is found, the execution continues from this point. There are some minor oscillations in the velocity controller when approach an object. However these are not critical at all, as long as the amplitude stays low ($\pm 20\%$ around v_{approach}).

The jerk in the experimental trajectory of experiment one and four (Section 5.1 and 5.4) seems to be very high, especially compared to jerk of the theoretical trajectory. However, the experimental jerk is still quite low when comparing it to the values in Figure 22 (center panel). It might be that jerk values in the order of magnitude of the ones of the theoretical trajectory are practically hard to achieve. However, also a jitter in the ROS timestamps can be considered as possible source of error.

Nevertheless, a position deviation occurs and accumulates each time the contact is lost. This deviation caused high force peaks and might also be problematic in other scenarios, in which an accurate position matters more. Therefore, this is a problem that needs to be solved.

In addition to that, the exertion of forces is not yet optimal. At some points, the deviation from the desired force was not within acceptable ranges. Here, more research is required. Different controllers with varying parameters must be compared and evaluated. This also requires more experiments with different stiffness of the object and the tool.

More experiments are in addition to the conducted ones needed for the further evaluation of the DMP framework. An important aspect here is the support of cyclic motions. This feature has been implemented, but yet only tested very roughly.

Noticeable during the recording phase was the difficulty of teaching accurately the desired force. Although the robot is actively compliant, the force needed to move the robot is high enough to prevent very finely tuned force profiles. Also the difference in joint configuration compared to a human hinders the demonstration of natural arm movements.

6 Conclusion

The objectives of this thesis were threefold. First, a survey and evaluation of current PbD algorithms was necessary in order to pick the most appropriate methods for the further research. Next, a software framework should be developed that is flexible and implements the chosen algorithm. Finally, the conclusion of the work was the validation of the software and hardware system by the development of in-contact task algorithms, thus extending the chosen PbD approach. All of these targets were met.

Initially, the PbD approach most appropriate for this work is selected. For that, first, current learning algorithms are reviewed. After that,the most relevant of these approaches are evaluated. The discussion leads to the conclusion that DMP is the framework of choice for this thesis. Throughout this work, no opposing results were found, which would have led to a different choice. Nevertheless, that does not mean that other frameworks, especially GMR, would not have been sophisticated enough as well.

Following this, the developed software framework is presented. The design decisions of the architecture are motivated. Next to that, the principle ideas of the components and their interactions are explained. In addition, some noteworthy implementation details are shown, especially concerning the use of unit quaternions. The critical discussion about the software architecture leads to the conclusion that the flexibility in the design eased and sped up the later development process. On the other hand, some issues need refinements, such as the interface between ROS and Orocos.

The biggest part of the thesis is about the novel algorithms for in-contact tasks. These algorithms are derived and explained. Some of these rely heavily on the capabilities provided by the controllers of the KRC. The perturbation handler uses the canonical system to slow down or stop the execution, in case of deviations to the nominal trajectory. The velocity controller is used to approach an object of desired contact. In the same section, also the force controller is presented. This controller regulates the force commands to the KRC and compensates for its imperfect measurements, in order to exert a desired force.

All these additions to the basic DMP framework are tested in the experiments. The results of the experiments are evaluated in the discussion thereafter. The general finding is that all extensions provide satisfactory results. The algorithms are able to handle rather complex PbD in-contact tasks, as the writing experiments demonstrate. Nevertheless, some problems are existing, which need fine-tuning of the code or partially even modifications of the approach. The former ones include for example the wrong positional offset, which is stored when the contact is lost. The latter include the bundling of the learning and execution phase. More investigation is needed for the force controller, where the biggest deviations occur.

Not much literature can be found about PbD for in-contact tasks. However, the approach followed in [11] is similar to the one presented in this work. While in this thesis, forces in the teaching phase are directly measured using a F/T-sensor at the tip of the arm, in [11] an external haptic device is used. In contrast to this work,

the 3 D force vector is taught and applied in the execution phase. In addition, the stiffness of the system is variable throughout the execution and depending on the variations across the different demonstrations. One drawback of their implementation is the handling of changes in the position of objects. The exerted force is higher than desired, when the object for the in-contact task is further away than expected and higher when closer. Thus on the one side, the approach of [11] is more sophisticated, on the other side it requires a static environment.

The mentioned results can also be put in context of the attributes of robots, which the introduction claimed to improve. These attributes were sensitivity, safety and flexibility. Most robots greatly lack each of them.

Sensitivity is especially introduced by the choice of hardware. The KUKA LWR has a torque sensor in each of its joints, giving it limited tactile sense, ordinary industrial robots do not have. The sensors are used in combination with the accurate dynamic model of the arm, allowing it to distinguish between internal and external forces. In this thesis, this is further improved by the F/T-sensor, which is attached to the tip of the arm. With this sensor, forces and torques can directly be measured at the tool.

Safety is improved by two main contributing components. The first are the torque sensor together with the dynamic model, which allow to limit the amount of exerted force. The second is the novel perturbation handler, which causes a movement to stop in the case of disturbances.

Flexibility is provided by three elements. First of all, by the use of the DMP framework. This framework is amongst the most flexible ones of all PbD approaches. Noteworthy here are for example the introduction of canonical terms, which allow online modifications of a trajectory. Second, the designed software framework does not restrict this flexibility, even more extends it. This was in detail described in Section 4.6. Lastly, PbD is inherently flexible. It leaves it up to the user to teach a robot all kinds of tasks.

Nevertheless, all these promising results must be seen within a bigger picture. World knowledge was also mentioned in the introduction as lacking attribute of robots. This is neither changed by the use of DMP nor by any of the other PbD approaches working on the trajectory level. Yet, symbolic learning might be able to improve the situation a little.

Thus, the approach that has been followed in this thesis is only one part of the puzzle. This approach needs a higher-level logic, especially a component conceiving the world and able to reason about action. This higher-level logic then defines tasks and makes use of DMP and similar methods to execute these tasks. The definition of a task may consist first of the choice of the movement (which is selected from a library of learned movements). This movement could then be further refined by adjusting the goal, the scale and the execution speed.

However, not all tasks fit in the scheme of predefined movements. One counter-example may be the relatively easy task of ironing. For this task, a certain trajectory or force profile is not that relevant. Yet, the focus is more on the coverage of the whole piece of clothing, while not creating any creases. The task highly depends on the geometry and the material. In addition, constant visual feedback is required to

check for undesired creases.

6.1 Future work

The possible future extensions to the developed system are numerous. In this following section, some ideas for interesting future steps are given.

Software architecture In a first step, the detected error and existing issues with the software framework should of course be fixed (see Section 4.7). Another improvement would be the possibility to concatenate movements. That means that the software is able to smoothly switch between different movements to achieve more complex tasks. Also the extensions discussed in the following section will require further adaptions to the software.

Dynamic Movement Primitives The previously described concatenation probably also requires changes in the DMP method or at least its implementation. It should be possible to start a learned motion from arbitrary points. This means that trajectories should be relative to the starting point and not fixed in space.

Currently, only force exertion in the Z direction are possible. The generalization to all directions should be rather simple. Also the exertion of torques could be implemented with the presented approach.

The experiments (Section 5) show that the current version of the perturbation handler slows down the execution. This is not desired and requires changes in the algorithm. Another version of the perturbation handler has already been designed, but neither tested nor implemented. This version takes into consideration the change of the error in the trajectory (the derivative over time). If the rate of change is negative (the error decreases), then the canonical system is not slowed down.

Another improvement would be variable stiffness, as described in [32] and demonstrated in [11]. If there is a big variance throughout different demonstrations for parts of the trajectory, then the stiffness in the execution phase for this parts is reduced. The interpretation of this variance is that the exact position is not relevant. Accordingly, the stiffness would be set to higher values, when the demonstrations do not show high variances.

General ideas In Section 5.6, it was described that the kinesthetic teaching approach is partially not optimal for the achievement of natural trajectories. This is due to the different embodiment and the difficulty to move the robotic arm in the way the human arm would move. There are (at least) two options to come around this situation. The first idea could be called iterative teaching. A trajectory is demonstrated, learned and executed as it is done now. During the execution phase, the perturbation handler is deactivated and the stiffness is set to a low value. This allows the demonstrator to alter the trajectory. The modified trajectory is recorded and taken as new input for the next iteration. This can be repeated several times until the resulting trajectory is satisfactory. A second approach is to use observational learning instead of kinesthetic teaching (see Section 2.1). This would require new

hardware (some kind of tracking system) and introduces the problem of different embodiment.

RL could help to further improve a learned trajectory. In this case, the user gives a basic demonstration of the movement and supplies a reward function. This reward function defines the optimization criteria of the movement (see Section 2.6). Then the robot tries to maximize the return of the reward function and thereby learns by itself.

In order to allow the robot to do more human like tasks, a hand is needed. A hand allows for much more versatile tasks. A hand is already existing for the arm. Yet, the hand needs to be integrated with the system and the software architecture. As the fingers of the hand are not backdrivable, one must think of an alternative to teach finger movements.

In the previous conclusion (Section 6), it was already stated that DMP is only one piece of the puzzle. The system developed in this thesis needs to be integrated in a higher-level logic. A first step could be the addition of an external camera. A camera (together with some additional software) would allow for example to determine a target. Applied to the writing experiment, the camera would recognize the position of the notepad and accordingly set this as goal point for the approach of the pen. As a step in the more distant future, symbolic learning could be introduced. Together with additional world-knowledge, the robot could eventually reason about its tasks and act according to a given goal, not to a defined movement.

References

[1] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot Programming by Demonstration. In Bruno Siciliano and Oussama Khatib, editors, *Handbook of Robotics*, pages 1371–1394. Springer, 2008.

[2] A. Montebelli and V. Kyrki. Transferring physical skills from humans to robots: Multimodal programming by demonstration for in-contact tasks. In *Workshop: Cognitive Robotics Systems: Replicating Human Actions and Activities*, 2013.

[3] B. Sciavicco, L. Sciavicco, V. Luigi, and G. Oriolo. *Robotics: modelling, planning and control*, chapter Force Control, pages 363–405. Springer, 2009.

[4] M. K. Vukobratović and V. Potkonjak. Dynamics of contact tasks in robotics. part i: general model of robot interacting with environment. *Mechanism and machine theory*, 34(6):923–942, 1999. doi: 10.1016/S0094-114X(97)00091-8.

[5] Y. Maeda, K. Oda, and S. Makita. Analysis of indeterminate contact forces in robotic grasping and contact tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1570–1575, 2007. doi: 10.1109/IROS.2007.4399022.

[6] D. J. Balkcom and J. C. Trinkle. Computing wrench cones for planar rigid body contact tasks. *The International Journal of Robotics Research*, 21(12):1053–1066, 2002. doi: 10.1177/0278364902021012003.

[7] G. A. Pratt, M. M. Williamson, P. Dillworth, J. Pratt, and A. Wright. Stiffness isn't everything. In *Experimental Robotics IV*, pages 253–262. Springer, 1997. doi: 10.1007/BFb0035216.

[8] T. Tsuji and Y. Tanaka. Bio-mimetic impedance control of robotic manipulator for dynamic contact tasks. *Robotics and Autonomous Systems*, 56(4):306–316, Apr 2008. ISSN 0921-8890. doi: 10.1016/j.robot.2007.09.001.

[9] Y. Zhu and E. J. Barth. Impedance control of a pneumatic actuator for contact tasks. In *IEEE Conference on Robotics and Automation (ICRA)*, pages 987–992, Apr 2005. doi: 10.1109/ROBOT.2005.1570245.

[10] B. Kim, J. Park, S. Park, and S. Kang. Impedance learning for robotic contact tasks using natural actor-critic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 40(2):433–443, 2010. doi: 10.1109/TSMCB.2009.2026289.

[11] P. Kormushev, S. Calinon, and D. G. Caldwell. Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *Advanced robotics: the international journal of the Robotics Society of Japan*, 25 (5):581–603, 2011. doi: 10.1163/016918611X558261.

[12] P. Kormushev, S. Calinon, and D. G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013. doi: 10.3390/robotics2030122.

[13] R. Bischoff, J. Kurth, G. Schreiber, R. Koeppe, A. Albu-Schaeffer, A. Beyer, O. Eiberger, S. Haddadin, A. Stemmer, G. Grunwald, and G. Hirzinger. The KUKA-DLR lightweight robot arm-a new reference platform for robotics research and manufacturing. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–8, 2010. doi: 978-3-8007-3273-9.

[14] S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 358–363. IEEE, 2006.

[15] V. Krüger, D. Kragic, A. Ude, and C. Geib. The meaning of action: a review on action recognition and mapping. *Advanced Robotics*, 21(13):1473–1501, 2007.

[16] A. Ude. Trajectory generation from noisy positions of object features for teaching robot paths. *Robotics and Autonomous Systems*, 11(2):113–127, 1993.

[17] J. Hwang, R. C. Arkin, and D. Kwon. Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1444–1449. IEEE, 2003.

[18] J. Aleotti and S. Caselli. Robust trajectory learning and approximation for robot programming by demonstration. *Robotics and Autonomous Systems*, 54 (5):409–413, 2006.

[19] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996. URL `http://arxiv.org/abs/cs/9603104`.

[20] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11:11–73, 1997.

[21] S. Schaal, C. G. Atkeson, and S. Vijayakumar. Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17(1): 49–60, 2002.

[22] C. E. Rasmussen. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[23] D. Nguyen-Tuong and J. Peters. Local gaussian process regression for real-time model-based robot control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 380–385. IEEE, 2008.

[24] S. Park, S. K. Mustafa, and K. Shimada. Learning-based robot control with localized sparse online Gaussian process. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1202–1207. IEEE, 2013.

[25] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(2):286–298, 2007.

[26] M. Muhlig, M. Gienger, S. Hellbach, J. J. Steil, and C. Goerick. Task-level imitation learning using variance-based movement optimization. In *IEEE Conference on Robotics and Automation (ICRA)*, pages 1177–1184, 2009.

[27] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard. Learning and reproduction of gestures by imitation. *IEEE Robotics & Automation Magazine*, 17(2):44–54, 2010.

[28] M. Toussaint, M. Gienger, and C. Goerick. Optimization of sequential attractor-based movement for compact behaviour generation. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 122–129. IEEE, 2007.

[29] S. Khansari-Zadeh and A. Billard. Imitation learning of globally stable non-linear point-to-point robot motions using nonlinear programming. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2676–2683, Oct 2010. doi: 10.1109/IROS.2010.5651259.

[30] S. M. Khansari-Zadeh and A. Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5): 943–957, 2011. doi: 10.1109/TRO.2011.2159412.

[31] S. Schaal, P. Mohajerian, and A. Ijspeert. Dynamics systems vs. optimal control – a unifying view. *Progress in brain research*, 165(1):425–445, 2007.

[32] 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.

[33] 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.

[34] S. Calinon, I. Sardellitti, and D. G. Caldwell. Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 249–254. IEEE, 2010. doi: 10.1109/IROS.2010.5648931.

[35] S. Calinon, F. D'halluin, D. G. Caldwell, and A. G. Billard. Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 582–588. IEEE, 2009. doi: 10.1109/ICHR.2009.5379592.

[36] E. Oztop, M. Kawato, and M. A. Arbib. Mirror neurons and imitation: a computationally guided review. *Neural Networks*, 19(3):254–271, Apr 2006. doi: 10.1016/j.neunet.2006.02.002.

[37] E. Oztop, M. Kawato, and M. A. Arbib. Mirror neurons: functions, mechanisms and models. *Neuroscience Letters*, 540:43–55, Apr 2013. doi: 10.1016/j.neulet.2012.10.005.

[38] A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[39] G. Schreiber, A. Stemmer, and R. Bischoff. The Fast Research Interface for the KUKA Lightweight Robot. In *IEEE Conference on Robotics and Automation (ICRA)*, pages 15–21, 2010.

[40] A. Albu-Schäffer, S. Haddadin, C. Ott, A. Stemmer, T. Wimböck, and G. Hirzinger. The DLR lightweight robot: design and control concepts for robots in human environments. *Industrial Robot: An International Journal*, 34 (5):376–385, 2007. doi: 10.1108/01439910710774386.

[41] W. Chung, L. Fu, and S. Hsu. Motion Control. In Bruno Siciliano and Oussama Khatib, editors, *Handbook of Robotics*, pages 133–159. Springer, 2008.

[42] G. Boggia, P. Camarda, A. Grieco, and G. Zacheo. *A Real-Time Wireless Communication System Based on 802.11 MAC*, chapter 3, pages 51–76. InTech, Mar 2010. doi: 10.5772/9522.

[43] P. Soetens. *The Orocos Component Builder's Manual*. The Orocos Project, 2.6.0 edition, 2012. URL `http://www.orocos.org/stable/documentation/rtt/v2.x/doc-xml/orocos-components-manual.pdf`.

[44] Wikipedia. RTAI. `http://de.wikipedia.org/w/index.php?title=RTAI&oldid=130667538`, visited on 28 July 2014. Wikipedia article.

[45] J. C. Craig and G. B. Rollman. Somesthesis. *Annual Review of Psychology*, 50: 305–331, Feb 1999. doi: 10.1146/annurev.psych.50.1.305.

[46] G. Robles-De-La-Torre. The importance of the sense of touch in virtual and real environments. *IEEE Multimedia*, 13(3):24–30, 2006. doi: 10.1109/MMUL.2006.69.

[47] *Lightweight Robot 4+: Operating Instructions*. KUKA Roboter GmbH, Augsburg, Germany, ba lbr 4+ v6 en edition, Dec 2012.

[48] *Six-Axis Force/Torque Transducer*. ATI Industrial Automation, Apex, NC, USA, 9620-05-transducer section-17 edition, Apr 2013.

[49] *Network Force/Torque Sensor System: Installation and Operation Manual*. ATI Industrial Automation, Apex, NC, USA, 9620-05-net ft-10 edition, Feb 2013.

[50] The Orocos Project. Smarter control in robotics & automation! `http://orocos.org/`, visited on 28 July 2014. Official Orocos homepage.

[51] ROS.org. Powering the world's robots. `http://www.ros.org/`, visited on 28 July 2014. Official ROS homepage.

[52] H. Bruyninckx. Open robot control software: the OROCOS project. In *IEEE Conference on Robotics and Automation (ICRA)*, volume 3, pages 2523–2528. IEEE, 2001. ISBN 0-7803-6576-3. doi: 10.1109/ROBOT.2001.933002.

[53] H. Bruyninckx, P. Soetens, and B. Koninckx. The real-time motion control core of the Orocos project. In *IEEE Conference on Robotics and Automation (ICRA)*, volume 2, pages 2766–2771, Sept 2003. ISBN 0-7803-7736-2. doi: 10.1109/ROBOT.2003.1242011.

[54] S. Cousins. Exponential growth of ROS [ROS Topics]. *IEEE Robotics & Automation Magazine*, 18(1):19–20, 2011. doi: 10.1109/MRA.2010.940147.

[55] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009.

[56] S. Cousins, B. Gerkey, K. Conley, and W. Garage. Sharing software with ROS [ROS topics]. *Robotics & Automation Magazine, IEEE*, 17(12):12–14, 2010. doi: 10.1109/MRA.2010.940147.

[57] Katholieke Universiteit Leuven, division PMA. Kuka LBR user group: FRI interface. `http://git.mech.kuleuven.be/robotics/kuka_robot_hardware.git`, visited on 28 July 2014. GIT respository.

[58] Stanford University. Fast Research Interface Library. `http://cs.stanford.edu/people/tkr/fri/html/page__k_r_l_files.html`, visited on 28 July 2014. FRI library documentation and manual.

[59] Universität Bremen – Artificial Intelligence. Information about the KUKA lightweight robot. `http://toychest.ai.uni-bremen.de/wiki/projects:kuka_lwr`, visited on 28 July 2014. university project page.

[60] B. Sciavicco, L. Sciavicco, V. Luigi, and G. Oriolo. *Robotics: modelling, planning and control*, chapter Kinematics, pages 39–103. Springer, 2009.

A Class diagram

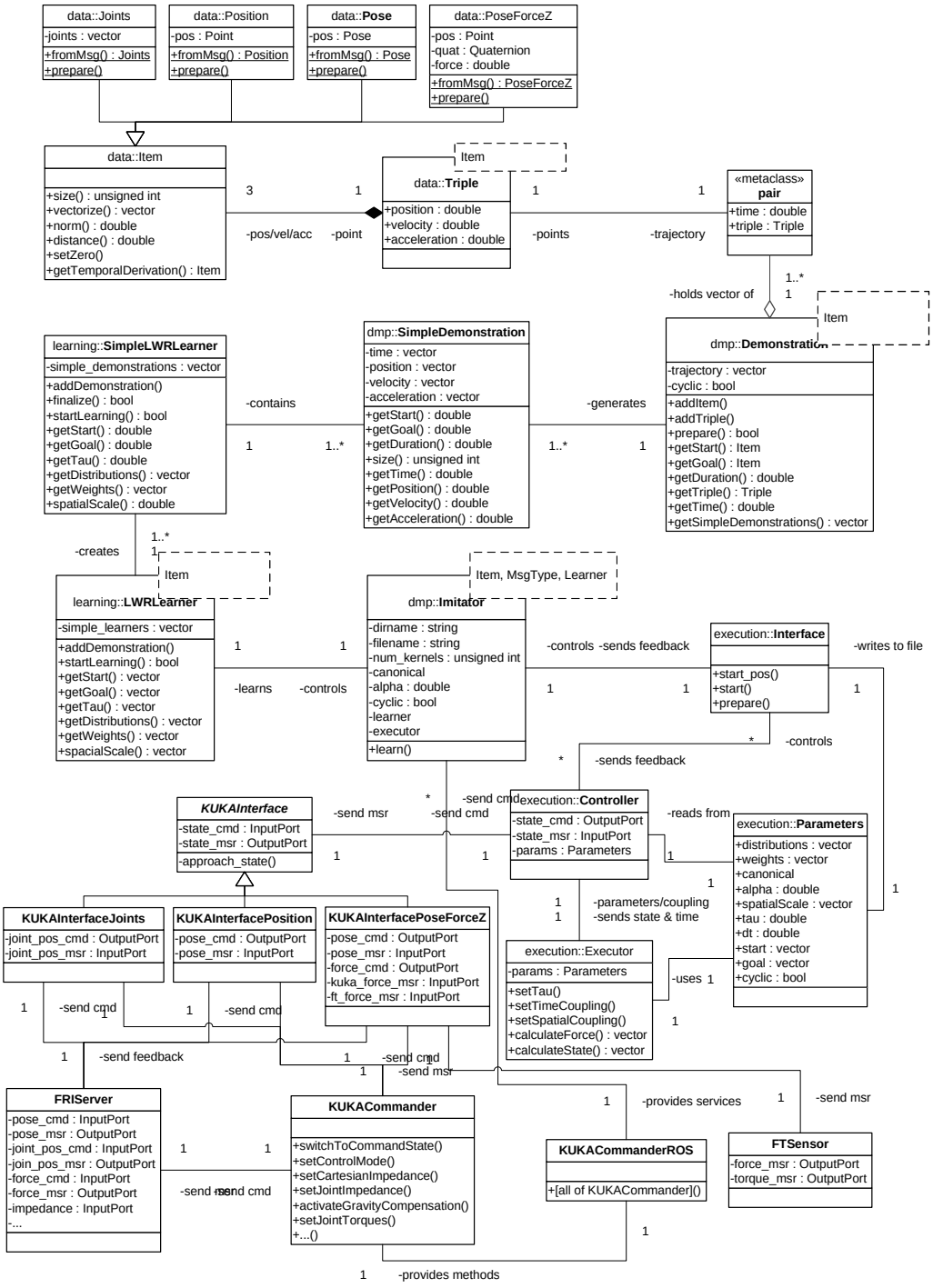

Figure A1: Class diagram of the most important classes, helper classes are not shown. Most of the public attributes and methods are displayed, but only few private ones.