Master Thesis

Adaptive Learning from Demonstration using Dynamic Movement Primitives

August 2012

Author: Glatz, Karl
Matriculation Number: 20570
Email Address: k@glatz.biz
Supervisors: Prof. Dr. W. Ertel
            Prof. Dr.-Ing. K. Wöllhaf
Eidesstattliche Versicherung


________________________
Karl Glatz
Programming a complex robot is difficult, time-consuming and expensive. Learning from Demonstration (LfD) is a promising methodology to avoid manual programming. Rather than explicit programming, a teacher demonstrates a task and the robot learns how to solve this task on its own. This thesis presents a Learning from Demonstration method for object manipulation tasks based on movement primitives.

A movement primitive encodes a small action such as grasping an object, while the combination of primitives can be used to describe a complex task. In order to represent the movements of a robot arm an approach called Dynamic Movement Primitives (DMPs) is used. Within the DMP framework a movement is represented by a set of differential equations. These equations describe the movement plan by calculating the next state \( \dot{x} \) based on the current state \( x \) (e.g. the end effector position). The parameters of the differential equations represent the shape of the movement and can be learned using a supervised learning algorithm. The DMP framework proposes the Locally Weighted Regression (LWR) algorithm in order to learn the parameters of the differential equation.

Demonstration of a task must be as straightforward as possible because even layman should be able to program the robot. This is achieved by an automatic decomposition of the task in small segments which can be represented by a movement primitive. The segmentation method analyzes the recorded sensor data of a demonstration for task specific segments.

An notable result of this thesis is a LfD method which provides online adaption to changes in the environment. Consider a human moves an object while the robot is performing a certain task, a standard movement plan is not able to adapt to such a change of the environment, which is one of the advantages of using dynamical systems to represent movements. Further, the learning from a single demonstration without any prior knowledge is an advantage over LfD methods which often require several demonstrations (at least 2). The representation of skills is kept as hardware independent as possible. The experiments show how learned skills can be transfered between two different robot arms.
3.3.4 Summary .................................................. 24
3.4 Conclusion .................................................. 25

4 Implementation Details & Experimental Results 29
  4.1 Technical Background ...................................... 29
    4.1.1 DMP Package ........................................... 30
    4.1.2 LfD Package ........................................... 30
    4.1.3 Robots .................................................. 31
  4.2 Experiments ................................................ 31
    4.2.1 Parameters ............................................. 32
    4.2.2 Experiment I: Coke in Cup ............................ 32
    4.2.3 Experiment II: Pouring Liquid ....................... 35

5 Conclusion & Outlook 37
  5.1 Conclusion ................................................ 37
  5.2 Outlook ................................................... 37

A Locally Weighted Regression 41

B Hierarchical Clustering 45

List of Figures 48

Bibliography 51
Chapter 1

Introduction

1.1 Motivation

In order to fulfill the dream of autonomous universal service robots operating in various life domains, robots need to be robust, affordable and flexible. The ability to acquire new skills is an important property of such a robot. Manual programming of all skills, as done for industrial robots, is not an option. Most end users are not willing to spend the time and effort to program the robot. Typically programmed behaviors are specific to a concrete situation. In industry, adaptation to changes in the environment is less important because the robots work in well-defined environments. Adaptation is however, essential for service robots for instance, which operate in the same space as humans.

The concept known as Learning from Demonstration (LfD) defines a promising methodology which enables robots to learn new skills in a human-like manner. A teacher demonstrates to the robot how to perform a task which in turn transforms this demonstration into an abstract skill.

1.2 Learning from Demonstration

In LfD a teacher demonstrates a task to a learner, in such a way that the learner is then able to reproduce the task. Using this simple interaction, even layman can program a robot. A human learner typically has difficulties reproducing certain behaviors exactly as demonstrated which is not true for machines. On the other hand, the adaption to changed environments or conditions comes naturally to humans, but is difficult for machines.

An overview and a categorical classification of LfD methods can be found in [ACVB09]. The Programming by Demonstration chapter of the Handbook of Robotics offers an introduction as well as a good historical overview ([BC07]). Distinction between the terms Learning from
**CHAPTER 1. INTRODUCTION**

*Demonstration* (LfD), *Programming by Demonstration* (PbD) and *Learning by Imitation* (LbI) is not of major interest in this thesis as these terms are typically used as synonyms in most literature.

### 1.2.1 History

Learning from Demonstration dates back to the 1980s. At this time robots became popular for industrial applications. The series production of robots resulted in a high demand for experts to program each different task explicitly. In order to reduce costs and programming time, simple teaching methods such as *teach-in* and *playback* were developed. In such scenarios, the teacher moves the robot arm manually or by teleoperation and a sequence of exact positions and orientations are recorded. The robot performs the movement exactly as recorded. These methods though successful for some tasks, suffered from drawbacks. For instance, this kind of imitation is only suitable for predefined work-spaces and processes. Nowadays, overcoming this limitation so as to enable robots the operation in unstructured environments such as households is a main topic of LfD research. Further, LfD prevents robots from learning skills from scratch, which is another reason why LfD is useful. Learning from scratch based on experience is known as *Reinforcement Learning*. While also a promising field of research it is impractical for many real-world manipulation tasks because of the high degrees-of-freedom of manipulators (known as the “curse of dimensionality”).

Instead of just relying on the memorization of a movement, LfD offers a more abstract view of tasks. This view is inspired by humans and animals learning. Although an appealing idea, it is difficult to determine which information is important to accomplish a task. This leads directly to the following key topics of LfD.

### 1.2.2 Key Topics of LfD

According to [ND01, BC12], the four key questions of LfD are:

“*What to imitate?*, “*How to imitate?*, “*When to imitate?*” and “*Whom to imitate?*”

Research in the field of LfD typically addresses only the first two questions. The remaining two questions are solved by using simple pragmatic solutions which have been proven to work for laboratory conditions.

**What to imitate?**

What to imitate is one of the most important questions in LfD. It refers to the problem of determining which aspects of a demonstration should be imitated. Depending on the task, some of the observed properties may be irrelevant for successful task execution. While
1.2. LEARNING FROM DEMONSTRATION

pouring water from a bottle into a cup for example, if a teacher approaches the cup from the left, is it necessary that the robot also approaches the cup from the left side? Or is it just that the pouring is important, no matter how it is being performed? This is related to the problem of distinguishing between noise and signal.

How to imitate?

How should the robot perform the parts of the demonstration that should be imitated? For example, if the teacher uses a foot to move an object, is it acceptable for a wheeled robot to bump it, or should it use a gripper instead? The efforts required to solve this question highly depend on the chosen demonstration interface.

1.2.3 Demonstration Interface

The interface between teacher and learner influences how the information is acquired. Different types of interfaces can be grouped into the following two categories:

**Imitation:** The teacher performs the task on his own. The data is obtained either by sensors attached to the teacher or observed by external sensors (like a camera). The observed data doesn’t match with the actions of the robot. Therefore, a mapping between the different embodiments must be found. This leads to the correspondence problem \cite{ND02} of physically non-identical platforms.

**Demonstration:** The teacher teleoperates the robot or moves the robot limbs physically (kinesthetic teaching). All joint states of the robot are recorded using sensors attached to the joints. This results in good data quality and is not affected by the correspondence problem. In both cases, teleoperation as well as kinesthetic teaching, a humanoid robot with dozens of joints is difficult to handle. The costs of building a proper demonstration setup are much lower than with the above described imitation techniques.

1.2.4 How to Learn and Represent Robot Skills?

Research in the topic of Learning from Demonstration results in different views on the learning and the representation of tasks. The most differing of the representations among learned robotic skills are symbolic encoding (high-level) and trajectory encoding (low-level). Symbolic encoding describes the task with a sequence of already known action primitives, like move-to-object. The trajectory encoding represents a task as a sequence of sensor data such as positions, velocities and accelerations.

A symbolic high-level LfD approach which learns concepts is presented in \cite{CE11, CE12}. The learning method proposed in these publications focuses on extracting an abstract de-
scription of the task from multiple demonstrations. Such an abstract task could be something like “stack all green cubes”. The actual movements are generated using a trajectory planner.

![Figure 1.1: Generalizing over multiple demonstrations (task space coordinates in mm)](image)

In contrast to this, successful work using a parametrized trajectory to encode the whole task has been done in \cite{SE10, Sch09} and \cite{CGB06, CGB07}. Both approaches use a regression method to generalize over multiple demonstrations (see Figure 1.1). They do this in both the joint space (positions of the joints) and the 3D task space (Cartesian coordinates calculated using robot kinematics). At task reproduction, several constraints such as the distance to the objects of the scene, are respected. The reproduction process generates a whole position-only trajectory which is executed using a standard controller. Thus interaction with the environment is impossible, feedback at execution time can not be considered, for example moved objects.

A more low-level view on learning movements is presented in \cite{INS03, SPNI04}. These works use dynamical systems to encode a movement. The main advantage of dynamical systems is their adaptiveness. In such systems the next state is always calculated from the current state by applying a fixed rule. Regression methods are used to approximate the non-linear part of these dynamical systems. Learning is possible with a single demonstrated trajectory \cite{SPNI04} or by generalization over multiple \cite{HGCB08}. Improving a trajectory with reinforcement learning has been done successfully (for example in \cite{PS08}). This encoding of a trajectory is point attractive, which means a trajectory is represented in terms of a fixed start and end (goal) state (see Figure 1.2). Unfortunately, this limits the field of application to tasks with a predefined goal position. In order to overcome this limitation it is possible to build a sequence of these movement primitives to represent more complex tasks. This bases on the assumption that complex movements can be described by small units of action \cite{Sch99}. 
1.3 Objective

The goal of this thesis is the definition and implementation of an adaptive Learning from Demonstration method for object manipulation tasks (e.g. pick and place objects). This method should be able to adapt to changes in the environment - even while performing a task. The learned tasks should be encoded hardware-independent, such that it is easy to provide an implementation for a different robot without learning the tasks again.

1.4 Outline

Chapter 2 introduces Dynamic Movement Primitives, a general way to learn and encode movements. The original formulation as well as an optimized version is described in this chapter. An explanation of the learning and execution process is provided, followed by a discussion on how to use the DMPs in the context of LfD.

Chapter 3 presents the proposed Learning from Demonstration approach, first describing the general LfD process, followed by detailed description of the sensor data based segmentation method.
Chapter 4 describes the experimental results of the proposed method using a real robot to demonstrate a task and reproducing the learned task in simulation using a different robot. A summary of the results as well as an outline of further work can be found in Chapter 5.
Chapter 2

Representation of Movements using Dynamical Systems

This chapter introduces the reader to the encoding of movements using dynamical systems, namely the framework known as Dynamic Movement Primitives (DMPs). Furthermore the application of this representation in the context of Learning from Demonstration is discussed.

2.1 Introduction

Nonlinear dynamic systems can be used to model discrete and rhythmic movements. This thesis focuses on discrete movements because they are more suitable for object manipulation tasks. The resulting control policies (CPs) are point attractive, which means that it is guaranteed they will reach the desired goal position. Due to the properties of dynamic systems, the goal position can be adjusted at any time. To put it simply, this kind of representation describes “how to move from a to b” whereas a and b are given as absolute positions.

A popular framework for the representation and learning of movements with dynamical systems is known as Dynamic Movement Primitives (DMPs), originally introduced in \[INS03\]. This chapter starts with an explanation of the original formulation. Later an improved version is described. The improvements are mainly for generalization in 3D task space and have been proposed in \[HPS08, HPPS09, PHS08, PHAS09\].
2.2 Dynamic Movement Primitives

A one-dimensional movement is generated by integrating the following set of differential equations, following the notation of [HPPS09].

\[
\begin{align*}
\tau \dot{v} &= K(g - x) - Dv + (g - x_0)f \\
\tau \dot{x} &= v.
\end{align*}
\]  

(2.1)

The first equation can be seen as a linear spring which is perturbed by an external force. The variables \( x, v, \dot{v} \) are the position, velocity and acceleration. \( \tau \) is a temporal scaling factor. \( K \) is the spring constant, \( D \) the damping term which is chosen such that the system is critically damped\(^1\), \( g \) is the goal position, \( x_0 \) is the start position, \( f \) is a non-linear function (the external force) which can be chosen to represent any desired movement. This set of equations is called transformation system.

For \( f = 0 \) and appropriate settings of \( K \) and \( D \) this equation converges to the the global attractor point \( g \) (Figure 2.1). In order to allow more complex trajectories on the way to the goal, \( f \) can be defined as a time-dependent, non-linear function. This leads to a non-linear dynamical system which may be difficult to solve, depending on \( f \). This problem is addressed by the introduction of an additional differential equation called canonical system ([SPNI04]),

\[
\tau \dot{s} = -\alpha s
\]  

(2.2)

with the non-linear function \( f \).

\(^{1}\)critically damped: the system converges to the goal as fast as possible without oscillating
2.2. DYNAMIC MOVEMENT PRIMITIVES

\[ f(s) = \sum_i w_i \psi_i(s) \]

where \( \psi(s) = exp(-h_i(s - c_i)^2) \) is the squared exponential basis functions with width \( h_i \), center \( c_i \) and adjustable weights \( w_i \).

The canonical system is a first order dynamical system with the same temporal scaling factor \( \tau \), as in Equation 2.1 \( \alpha \) is a predefined constant. The phase variable \( s \) is initialized with one \( (s(0) = 1) \). The constant \( \alpha \) is chosen such that the phase variable \( s \) goes from one to zero as time passes \( (s \rightarrow 0) \). The canonical system (Equation 2.2) can be solved symbolically, its solution being \( s(t) = exp(-2t) \) as depicted in Figure 2.2.

The boundedness of \( f \) (assuming bounded weights \( w_i \)) and a decreasing \( s \) ensure that the influence of the function \( f \) decreases towards the end of the movement and the equation globally converges to the unique attractor point \( g \). As the function \( f \) is only dependent on the phase variable \( s \), and not directly on time, it is possible to change the movement duration by adjusting \( \tau \).

2.2.1 Learning DMPs

Learning from single demonstration (one-shot learning) is one of the favorable characteristics of a DMP. Learning the path from start to goal position is equivalent to finding the appropriate weights \( w_i \) of \( f \). Considering the definition of DMPs given in Section 2.2 learning function \( f \) can formulated as a supervised learning problem. The target values are defined by the following function,

\[ f_{\text{target}} = \frac{-K(g - y) + D\dot{y} + \tau \ddot{y}}{g - x_0}. \]  \( \text{(2.4)} \)

Given a recorded trajectory with the arrays \( y, \dot{y}, \ddot{y} \) (position, velocity and acceleration at time \( t \)) with a total of \( T \) elements, in order to get \( f_{\text{target}} \), just replace \( x, v, \dot{v} \) with \( y, \dot{y}, \ddot{y} \) and solve Equation 2.1 for \( f \). The start position \( x_0 \) is given by \( y(0) \) (first element of \( y \)), and the goal position \( g \) by \( y(T) \) (last element of \( y \)). The input for \( f_{\text{target}} \) is generated by evaluating \( s(t) \). The resulting training samples are given by \( (s, f_{\text{target}}) \). If the recorded trajectory only contains positions, the corresponding velocity and acceleration values can be calculated.

\( s_{\text{min}} = 0.001, \alpha = -\log(s_{\text{min}})/\tau \)
The weights $w_i$ of $f$ can be determined by minimizing the error criterion $J = \sum_i (f_{\text{target}}(s) - f(s))^2$ which is a linear regression problem. A modified version of Locally Weighted Regression (LWR) provides a fast and suitable method to approximate $f$. This method fits $n = |w|$ local linear models (see third image in Figure 2.3). All data points are weighted by the kernel functions $\psi_i$. A straight line is fitted to the weighted points. At prediction all linear local models are evaluated and weighted by the respective kernel function $\psi_i$. A more detailed description of the algorithm is attached in Appendix A.

Assuming a fixed number of kernels (learning parameters) for $f$ limits the length and complexity of the learned movement. Such a limitation is not a serious problem because a single DMP is meant to be a small unit of action. The results of the approximation with $|w| = 20$ should be still good enough for short movements.

Figure 2.3: (1) LWR learned target function $f_{\text{predicted}}$ (green dashed) and the actual $f_{\text{target}}$ (blue); (2) 8 Gaussian weight functions, $|w| = 8$; (3) Local linear models

Figure 2.4: Position, velocity and acceleration of a minimum jerk trajectory (start: 0.3, goal: 1.4) (blue); DMP movement with learned function $f$ plotted in Figure 2.3 setup with initial start and goal (green dashed curve).
2.2. EXECUTING DMPs

In order to execute a DMP both the transformation system and the canonical system have to be integrated. The learned function \( f_{\text{predicted}}(s) \) is driven by the canonical system \( s \) and modifies the default behavior of the transformation system. The movement can be adjusted by setting the parameters \( x_0 \) and \( g \) (start and goal position) as well as the temporal scaling parameter \( \tau \). These parameters can be chosen as required by the task (Figure 2.5).

![Figure 2.5: Sketch of a one-dimensional DMP (PHAS09)](image)

2.2.3  CHARACTERISTICS OF DMPs

**Multiple Degrees of Freedom** Using DMPs for real-world applications like moving a robotic arm requires them to encode more than one-dimensional movements. This can be achieved by using one transformation system for each dimension of the data.
The canonical system is shared across all transformation systems (see Figure 2.7).

\[
\tau \dot{s} = -\alpha s
\]

Figure 2.7: Sketch of a multi dimensional DMP.

**Sequence DMPs** As stated earlier in this chapter, DMPs are intended to be single basic units of action. To allow more complex movements it is possible to sequence several DMPs.

**Movement Recognition** The similarity of two movements can be determined by comparing the weights \( w_i \) of two movements with each other.

**Online Adaption** The goal position \( g \) in Equation 2.1 can be changed at any time of the execution. In the LfD context, online adaptation is the crucial advantage of using DMPs to represent a movement.

### 2.3 Modified DMP Formulation

The original formulation, presented in the last section, has some numerical problems with the adaption to new goals (illustrated in Figure 2.8).

The work in [HPPS09] describes a modified DMP formulation which is not affected by this drawbacks. While the transformation system is changed to the following equation, the canonical system stays the same as in equation 2.2.

\[
\begin{align*}
\tau \dot{v} & = K(g - x) - Dv - K(g - x_0)s + Kf(s) \\
\tau \dot{x} & = v.
\end{align*}
\]

The function \( f(s) \) is the same as defined in the original formulation (Equation 2.3). Note, the function \( f \) is not multiplied by \((g - x_0)\) anymore. This helps to prevent problems with
2.3. MODIFIED DMP FORMULATION

Figure 2.8: Drawbacks of the original DMP formulation. In the left Figure the sign of $(g - x_0)$ has changed and the movement is mirrored (red line). The changes of the goal in the right Figure are extremely small (-0.02), but the movement between start and goal position doesn’t adapt as expected.

movements which start and end at the same position $(g = x_0)$. It also makes the adaption to only slightly changed goals more optimal (compare Figure 2.8 with 2.9). The most important characteristic of this improved formulation is that it generalizes to new goals nicely even if the sign of the term $(g - x_0)$ changes compared to $(g_{new} - x_0)$. The third term $K(g - x_0)s$ is required to avoid jumps at the beginning of the movement ([PHAS09]).

The target function $f_{target}$ is generated as previously described by solving Equation 2.5 for $f$,

$$f_{target} = \frac{\tau \ddot{y} - D \dot{y}}{K} + (g - y) - (g - x_0)s. \quad (2.6)$$

Figure 2.9: Adaption to new goals with the modified DMP version. Compared to Figure 2.8 this is a huge improvement in adaption accuracy.
Due to its clear improvements, this modified version is used in all following descriptions and experiments. A simple one-dimensional implementation of both versions is available online.

2.4 Movement Primitives in the Context of LfD

The previous sections have described how a movement is represented, how it can be learned and how it is executed. It should be clear that DMPs are useful to represent movements with a known target position. The following section discusses the application of DMPs in the context of Learning from Demonstration for object manipulation tasks. A typical object manipulation task consists of multiple actions with different target positions.

2.4.1 Movement Library

Building a library of movements in order to use them as building blocks is one method to organize DMPs. The idea of movement (or motion/skill) libraries is simple. A teacher demonstrates, then labels each small movement and stores them in the library. Each movement represents a useful simple building block such as grasp-object.

A high-level reasoning layer can use this library to build a sequence of movements for a task. One theoretical approach to build such a system, described in [GMP+06], is known as Object Action Complexes. Building a movement library requires a lot of teaching work, each single short movement must be demonstrated and labeled. The composition of these building blocks by a reasoning layer still requires information of a specific task. This information can either be manually programmed or gathered through demonstration by any of the known demonstration interfaces (see Section 1.2.3).

2.4.2 Movement Segmentation

In general, movement segmentation is the process of dividing a long trajectory into smaller parts (the segments). As a DMP is meant to be a small building block, the segmentation of a long movement is particularly important. In the following several different segmentation approaches are introduced.

Segmentation of a movement based on the approximation error of a linear dynamical system was proposed in [DK04]. Similar to this, the approximation error of DMPs can be used as segmentation criterion. In [Pas08], a movement is recursively segmented based on this error. The point at which the movement is split, is determined by the minimum velocity of the current segment. While these are simple and effective methods to segment

\[ \text{https://github.com/carlos22/pydmp} \]
a movement, they are not well suited for LfD. The key advantage of LfD is the adaption to changing environments. The only parameters of a DMP are start and target (goal) position (assuming a fixed duration $\tau$). Therefore an adaption can only happen at the split point between two segments, which requires split points that are related to the task.

The data of a movement library can be utilized to solve the segmentation problem by recognizing the pre-trained movements. This approach is common among the vision community. In [MTSS11] such an approach is used with the ability to reproduce movements on a robot. While this is an interesting view of the segmentation problem, it requires a movement library which contains good quality movement primitives. It is possible that the segmentation fails to find an appropriate movement primitive in the movement library. Furthermore, it is an open question if this approach is able to provide an applicable segmentation for object manipulation tasks.
Chapter 3

Learning from Demonstration Approach

This chapter describes the learning and reproduction of object manipulation tasks. A sequence of DMPs is used to generate the movements.

3.1 Introduction

LfD batch-learning approaches typically analyze multiple demonstrated trajectories for similarities in order to identify success-critical parts of a task (see [SE10]). This LfD approach is based on a single demonstrated trajectory, which makes it much more difficult to identify such success-critical parts.

Instead of focusing on similarities in multiple demonstrations, this work is built up on the identification of task-critical events. A task-critical event divides a complex task into a sequence of subtasks. An example for such a task critical event might be “grasping a cup”. Task-critical events are identified by analyzing different sensor data like object positions or the position of the end effector.

The movement of a subtask is generated by a single DMP. The adaption to changes in the environment is made possible by the ability to change the goal position of a DMP. In order to know how the goal position should be changed, a reference object must be identified. Tracking the position of this object enables the system to update the goal position online, which in turn makes it possible to move objects while a task is performed.

3.2 Overview

This section gives an overview of the LfD process. The following section is completely dedicated to the segmentation method.
3.2.1 Robot Abstraction

There are two common coordinate systems used to describe the position of a robot arm. In *joint space* each joint state defines a dimension. In *task space* 3D coordinates (position) and the orientation of the end effector are used to describe the position of the robot arm. Forward kinematic is a mapping from joint space to task space. The inverse mapping from task space to joint space is called inverse kinematics. Because it represents each joint of the robot arm, the joint space depends on the physical model of the robot.

The learned skills should be interchangeable between robots. In order to achieve this, the representation of the task needs to be independent of the hardware and the physics of the robot used for demonstration. The first step to achieve this is the encoding of the robot arm movements in the *task space* which unlike joint space is not dependent on the physical model.

In the past decades various types of end effector tools have been developed, mostly for industrial robotic purposes. The most common form is a grasping unit which again encloses endless variations (see Figure 3.1 for some examples of differently shaped grippers). Regardless of their diversity in shape and detailed functionality they all share one common feature which is the ability to grasp and release objects. While it is possible to use a couple of transformation systems to encode the movement of an end effector tool, the generated movement can only be used with the same or very similar hardware. Therefore, the transformation systems encode the movement in task space only (Figure 3.2), while the end effector tool action is represented with the two states, grasp (close) or release (open).

3.2.2 LfD Process

**Demonstration**

In order to demonstrate a movement, kinesthetic teaching is used (see 1.2.3). The human teacher manually moves the arm of the robot and the states of all joints are recorded. The initial position of the objects must be provided by some kind of object recognition.

**Learning a Task**

In the first step of the learning process the joint space trajectory is converted to a task space trajectory using a forward kinematics calculation. The pose of the objects and the Cartesian 3D task space trajectory are transformed into a common frame. The remaining
3.3. MOVEMENT SEGMENTATION BASED ON SENSOR DATA

As mentioned in Section 2.2.1, the length and complexity of a learned movement is limited by the number of learning parameters (i.e., the number of weights $w_i$ of the function $f$).
Given enough learning parameters (in the limit one for each point), DMPs are able to learn any complex movement. Even with a limited number of learning parameters, the DMP can still generate a reasonable movement. If the generated movement suits for the task depends on how important the exact reproduction of the learned shape of the movement is. Object manipulation tasks usually require a less exact reproduction of the shape than a task such as hitting a baseball.

A limited number of learning parameters is not the only reason why DMPs are considered as basic units of action. At least as important are the parameters of a DMP, the start position, the goal position and the duration of the movement. As the goal position can be changed at any time, DMPs fit perfectly for subtasks where each goal position is defined by an observable feature. Consider a typical object manipulation task like “pick up an object and put it in a box”, the actions of such a task could be, (1) approach the object, (2) grasp the object, (3) move the object over the box, and (4) release the object. Each action or subtask requires a specific goal position relative to one of the objects. Ideally, the decomposition of a demonstration is performed automatically without any guidance by the teacher.

One of the important characteristics of Learning from Demonstration is the ability to adapt to changes in the environment (see Section 1.2). Thus, any changes in the environment must be considered in the reproduction process. This segmentation method decomposes a demonstration into subtasks and defines a reference feature for each subtask.

### 3.3.1 Task-Critical Events

A task-critical event is defined as a point in time at which the current position of the robot arm is critical for the success of the task. The moment at which the robot arm is grasping an object, for example, is such a point in time because the grasping is only successful if
the robot arm is at the correct position.

For typical object manipulation tasks, the following three major types of task-critical events can be identified.

**Picking and Placing of Objects** The moment before the gripper grasps the object in the demonstration is a task-critical event. If the object position changes, the position before the grasp must be adapted to the new object position.

**Rotation of Objects** Rotation of objects is an indicator for a task-critical event. For example if a robot pours some liquid, it is required that the robot arm moves to the target before the actual pouring starts. Therefore the position at which pouring should begin is task-critical.

**Spatial Proximity to Objects** The proximity between the end effector and an object can also be considered as a task-critical event. For example, if the robot pushes a button, the distance to the button is important for the success of the task.

These events often occur together in one trajectory. Consider a pouring task where the contents of a can is poured into a cup. The can must first be grasped before it’s content can be poured out. Each event needs to refer to a subtask related item, in the pouring example, the first subtask related item would be the object “can”.

### 3.3.2 Analysis of Movement for Task-Critical Events

In order to identify the described task-critical events, the following set of sensor data is required:

- Demonstrated Trajectory
  - Cartesian 3D task space trajectory (position and orientation of the end effector over time)
  - Gripper movement (distance between two fingers over time)
- Initial position of objects (at the beginning of the demonstration)

**Grasp and Release of Objects**

Movement of the gripper can be represented as a one-dimensional trajectory \( \Gamma(t) \) with \( t = 0..N \) elements at a specific sampling rate. The velocity of the gripper movement is given by the first derivation of \( \Gamma \). The noisy raw data is smoothed using Gaussian
smoothing (also known as Gaussian blur) which is a well-known low-pass filter (green line in Figure 3.4) for all kinds of data, commonly used by image manipulation applications.

Searching for clusters in the time dimension of all points which are above or below a certain threshold is more robust than searching for all local maxima and minima. A cluster is a group of points which share a certain similarity – in this case temporal proximity. The definition of a minimum cluster size is used to ensure that the movement of the gripper had a certain duration.

As clustering algorithm the single-linkage version of the Hierarchical Clustering (Appendix B) is used. An advantage of this algorithm is that there is no prior knowledge of the number of clusters needed. The distance function \( d \) which defines the distance between two elements is set to \( d(x, y) = |x - y| \). How many clusters are found can be adjusted by defining a minimum distance \( D_{\text{min}} \) between clusters. The distance between each of the resulting clusters is at least \( D_{\text{min}} \). Applied to the given problem this is equal to the duration between two gripper events. A good choice of this parameter is something between 1.0 and 1.5 seconds.

The computational complexity of single-linkage clustering is \( \theta(n^2) \). If the data has a high sampling rate, like 1000 Hz, the runtime of the clustering algorithm is quite long. In order to minimize the runtime, the trajectory can be reduced by a factor \( n \), so only each \( n \)-th element is considered for clustering. In case of a high sampling rate, this does not affect the detection of events. The blue areas in Figure 3.4 represent a found cluster.

As mentioned earlier, the data of the demonstration only contains the initial position of the objects. This is an artificial limitation mainly due to the fact that the currently available real-world object detection is not able to track objects. The identification of grasp and release moments is used to update this data.

**Rotation of Objects: Changes in Roll Angle**

End effector orientation can be described by the Euler angles namely yaw, pitch and roll \((\Psi, \Omega, \Phi)\). The previously explained method to detect events of the gripper is also used to detect changes of the end effector roll angle. Data of the roll angle indicate a rotation of the currently manipulated object. Therefore only data points at which an object is actively manipulated are considered (magenta line in Figure 3.5).

In general, the yaw angle is also relevant for the rotation of an object and therefore should be also analyzed for task-critical events. But many robot arms represent the end effector roll angle by a single joint which is usually used to rotate objects. However, the analysis could easily be extended to consider the yaw angle as well. In Figure 3.5 the roll angle velocity of a simple pouring task is plotted.
3.3. MOVEMENT SEGMENTATION BASED ON SENSOR DATA

Figure 3.4: Velocity of the gripper fingers (blue), the green line is the Gaussian-smoothed version of the velocity signal. The ignored area (noise prone) is marked with an upper and a lower margin illustrated as gray horizontal lines. The blue areas indicate the object manipulation units found by clustering of all points lying outside of the ignored area. Red circles indicate a gripper release, cyan squares a grasp.

3.3.3 Identify Reference Objects

Define a Reference Object for each Subtask

The previously described task-critical events divide the task into subtasks with a goal position. This position is typically relative to an object. In order to find a reference object, the end effector position of the robot arm at time of the task-critical event is compared to the position of all objects. The distance between object position $o$ and end effector position $p$ is given by the euclidean distance,

$$d_{euc}(o,p) = \sqrt{\sum_{i=0}^{n} (o_i - p_i)^2}.$$ 

As a reference object, the object closest to the end effector is used. The nearest object $o_{nearest}$ is defined as the object with the minimum distance between the events end effector position and the object position. The currently manipulated object is always excluded,

$$o_{nearest} = \text{argmin}_{o \in \text{Objects}} d_{euc}(o,p).$$

The raw distance of the $x, y, z$ components between object and end effector is stored as it is required by the execution unit to adapt the goals of the movement correctly.
CHAPTER 3. LEARNING FROM DEMONSTRATION APPROACH

Figure 3.5: Velocity of the roll angle (blue), the green line is the Gaussian-smoothed version of the velocity signal. The ignored area (noise prone) is marked with an upper and a lower margin illustrated as gray horizontal lines. The blue areas indicate the event units found by clustering of all points lying outside of the ignored area. Red diamonds indicate actual event time. Times in which an object is manipulated is marked by a magenta line with arrows at both ends.

Spatial Proximity to Objects

Robot arm’s spatial proximity to an object can be relevant for successful task execution. The distance between the end effector and an object is used to discover such events. This distance is also very small if the object is grasped or released, which is an already identified task-critical event. An event is only valid, if there are no previously found events nearby the identified candidate. The definition of “nearby” is given by a parameter called minimum segment length. The minimum segment length is illustrated as a black dashed line in Figure 3.7. In time periods where an object is moved or in any other way manipulated, the spatial proximity between objects and the end effector is not considered as task-critical.

3.3.4 Summary

The complete segmentation process can be summarized as follows:

1. Discover task-critical events by changes in gripper states. Update object positions and remember at which times an object was manipulated. Find a reference object for all events by searching for the nearest object at time of the event.
2. Analyze changes in roll angle of the end effector to find object rotation events. Only consider time periods where an object is being manipulated. Find a reference object for all events by searching for the nearest object at the event time – exclude the currently manipulated object from this search.

3. Compare the positions of each object with the position of the end effector. Identify areas where the distance is very small. Find the point with the minimum distance in this area. Add this time to the task-critical events, if there is no other event nearby and the object is not currently being manipulated.

3.4 Conclusion

The presented LfD method allows the learning of complex object manipulation tasks. Compared to other LfD methods which require multiple demonstrations, the benefit for the teacher is clearly the single demonstration. However, one drawback of this method is that the resulting skill is more dependent on the quality of the demonstration. While all LfD methods suffer from the influence of the teacher on the quality of learned movements, a single demonstration is always more sensitive to misguidance by teachers.
CHAPTER 3. LEARNING FROM DEMONSTRATION APPROACH

Figure 3.7: Euclidean distance between objects and the position of the end effector. The black crosses represent previously identified task-critical events. Manipulation of an object is marked with a black dashed line. A valid task-critical event is marked with a green circle. Blue and red circles are rejected event candidates – blue means the minimum segment length would be violated by splitting at this time – red indicates that the object is manipulated at this time.

The described method has similarities with other symbolical subtask sequencing methods (like in [FMJ02]), with the difference that the subtasks are not explicitly labeled. Each task contains its individual actions (subtasks) which are not shared between tasks, because the movement of each subtask is considered individually.

An example of a complete task space trajectory, where one object is put into another object, segmented into subtasks with assigned reference objects is depicted in Figure 3.8.
Figure 3.8: Example of a segmented trajectory: (a) approach obj1, (b) grasp obj1, (c) move obj1 to obj2 and release obj1, (d) move to home position. Each red diamond represents a task-critical event labeled with a number indicating the reference object. The gripper action is written in square brackets (release/grasp).
Chapter 4

Implementation Details & Experimental Results

This chapter covers aspects of the implementation, describes the experimental setup and shows the results of two experiments.

4.1 Technical Background

The implementation is based on the Robot Operating System (ROS)\textsuperscript{1}. The core of ROS is a simple, yet powerful network-enabled inter process communication (IPC) framework. Unlike traditional operating systems it runs not directly on hardware, but on top of operating systems like Ubuntu Linux (which is the officially supported operating system).

In the ROS terminology, a program which participates in the communication is called a \textit{node}. Nodes use messages to communicate with each other. Such a message can be published to a \textit{topic}. A node can be subscribed to topics to receive all \textit{messages} published to this topic. While the communication using publish/subscribe happens asynchronously, \textit{services} offer synchronous communication between nodes. Each node can be configured using \textit{parameters} which are adjustable at runtime. All of these concepts are represented in a common \textit{naming scheme} inspired by the UNIX filesystem scheme. Furthermore a package and dependency management is provided. A \textit{package} is the smallest unit, containing one or more nodes. A composition of packages is called a \textit{stack}. Further introduction about the concepts of ROS can be found in \cite{QGC+09}.

On top of these basic concepts, ROS also provides several abstractions of robotics hardware. For example, the physics of a robot are described by a URDF (\textit{Unified Robot Description Format}) file which contains all joints and their properties. This simplifies the simulation of

\footnote{http://ros.org}
the robot and provides general interfaces for forward and inverse kinematics calculations. Dynamic transformations between robot frames (coordinate systems) are provided by the core package tf.

### 4.1.1 DMP Package

An implementation of Dynamic Movement Primitives is included in the ROS stack called “usc-clmc-ros-pkg” - a collection of stacks from the Computational Learning and Motor Control Lab of the University of Southern California.

The implementation of the DMPs itself is generally robot independent and the required robot properties can be defined as ROS parameters.

In order to execute the actual movement of a robot arm, the standard movement actions (like JointTrajectoryAction) are not suitable because they require a full movement plan. An additional step in order to create appropriate motor commands from the trajectory, generated by a DMP, is required. Such a DMP controller for Cartesian task space is included in the package for the PR2 robot.

As part of the implementation, the DMP package has received some minor modifications. The modified as well as the original packages are available on github[^2].

### 4.1.2 LfD Package

The dmp_lfd ROS package contains the following nodes:

- **learn_dmp_lfd_srv** provides the service learn_dmp_lfd which covers the whole learning process. The demonstration is requested as input message with the message type LfDDemonstration. The resulting task is written in a bag file[^3] containing the message DMPLfDTask. This package requires the forward kinematics service of the learning robot (usually published as /arm_namespace/get_fk).

- **execute_dmp_lfd_task** offers an action to execute a DMPLfDTask message. This message contains a bunch of DMPLfDTaskSegment messages which includes the learned DMP, the reference object and additional information like the gripper action.

- **lfd_track_object** is a node which advertises the action to track a specific object. At the moment this is achieved by using a fake object recognition which simply asks the simulation for the object position. This action is used by the execute_dmp_lfd_task.

[^2]: forked version with minor modifications: [https://github.com/carlos22/usc-clmc-ros-pkg](https://github.com/carlos22/usc-clmc-ros-pkg)
[^3]: bag files: serialization format for ROS messages
4.2. EXPERIMENTS

4.1.3 Robots

Demonstration Robot: Kate

The demonstration robot KATE is a proprietary robot. The main components of this robot are a Neuronics Katana arm and a Pioneer P3DX platform (see Figure 4.1). The Katana robot arm is a lightweight arm with 6 degrees of freedom (6-DOF). All software components are based on the robot communication framework SmartSoft\(^4\). The robot can be instructed by a speech recognition system. Details of the object recognition system can be found in [FSE12].

After a successful demonstration, all captured joint states of the robot, as well as the initial position of the objects are stored in an XML file. This file is converted to a ROS bag file using a script.

Reproducing Robot: Simulated PR2

The PR2 (PR stands for “personal robot”) is a full service robot developed by Willow Garage. ROS is the native software of the PR2. The software packages includes a simulation of the PR2 using the simulation environment Gazebo\(^5\). The PR2 is one of a few robots, where the ROS programming interface (actions and topics) of the simulated and actual hardware are widely compatible, even on a low level of control. The PR2 offers two 7-DOF arms. Because the teaching robot Kate only offers one arm, the right arm of the PR2 is used to reproduce the task.

4.2 Experiments

The following section presents the results of the LfD method applied to two common object manipulation tasks. Videos of the tasks show how the robot performs the task while some of the objects are moved to different locations. The result of the segmentation is depicted by three different graphs for each task.

\(^4\)http://smart-robotics.sourceforge.net/

\(^5\)http://gazebosim.org/
4.2.1 Parameters

Table 4.1 lists all the available parameters of the segmentation process including the values used in the experiments with the teaching robot Kate.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_frequency</td>
<td>Frequency of the recorded data.</td>
<td>1000 Hz</td>
</tr>
<tr>
<td>min_segment_duration</td>
<td>Minimum duration of a DMP movement.</td>
<td>1.0 s</td>
</tr>
<tr>
<td>gripper_change_min_duration</td>
<td>Minimum duration of a gripper event.</td>
<td>1.0 s</td>
</tr>
<tr>
<td>gripper_change_distance_duration</td>
<td>Minimum time between two gripper events.</td>
<td>1.5 s</td>
</tr>
<tr>
<td>roll_angle_change_min_duration</td>
<td>Minimum duration of a rotation event.</td>
<td>1.0 s</td>
</tr>
<tr>
<td>roll_angle_change_distance_duration</td>
<td>Minimum time between two rotation events.</td>
<td>1.0 s</td>
</tr>
<tr>
<td>object_distance_threshold</td>
<td>Maximum euclidean distance between object and end effector to be considered as event candidate.</td>
<td>0.13 m</td>
</tr>
<tr>
<td>roll_angle_change_threshold</td>
<td>Changes in roll angle, values below this threshold are considered as noise.</td>
<td>0.0002 rad</td>
</tr>
<tr>
<td>gripper_change_threshold</td>
<td>Changes in gripper distance (values below this threshold are considered as noise)</td>
<td>0.00005 rad</td>
</tr>
</tbody>
</table>

Table 4.1: Values and explanation of all parameters used in the segmentation process.

4.2.2 Experiment I: Coke in Cup

The coke in cup task consists of the following steps, grasp the coke can, put it near the cup, grasp it again and release it over the cup. The coke in cup task is certainly not a real-world task, but due to the limited number of available objects for simulation, it can be generalized as: put anything into a container.
4.2. EXPERIMENTS

Figure 4.2: Coke in cup task – reproduction in Gazebo with PR2 Robot.

The cup is moved to the left towards the end of the task execution (see Pictures 11 and 12 in Figure 4.2). The robot adapts the movement online to reach the cup. A full video of this task is available online.

Figure 4.3: 3D task space trajectories of the coke in cup task.

(a) Segmented demonstration trajectory. The red diamonds represent a task critical event. Each of the different colored trajectories represent a sub-task movement.

(b) Reproduced trajectory.

http://karl.glatz.biz/l/valfddmp1
(a) Gripper events. The blue areas are gripper actions. Red circles indicate a gripper release (open), cyan circles a gripper grasp (close).

(b) No rotation events could be identified because all values below the gray line (parameter roll_angle_change_threshold) are considered as noise. Magenta lines represent an object manipulation.

(c) Spatial proximity between the end effector and each object. Black arrows represent an object manipulation, black crosses a previous detected task-critical event. All candidates (blue circles) are not considered task-critical events because the distance of those events to the already identified events is too small (parameter min_segment_duration).

Figure 4.4: Task-critical events of the “coke in cup” task.
4.2. EXPERIMENTS

4.2.3 Experiment II: Pouring Liquid

Figure 4.5 shows the reproduction of the pouring task, a full video of this task is available online.

Figure 4.5: Pouring task - reproduction in Gazebo with PR2 Robot.

(a) Segmented demonstration trajectory. The red diamonds represent a task critical event. Each of the different colored trajectories represent a sub-task movement.

(b) Reproduced trajectory. The start position of the robot is different to the start position in the demonstration which results in an adapted movement (blue curve).

Figure 4.6: 3D task space trajectories of the pouring task

7http://karl.glatz.biz/l/valfddmp2
(a) Gripper events. Two gripper release (open) events marked with red circles, one gripper grasp (open) marked with a cyan circle.

(b) Two rotation events could be identified. Red diamonds denote the start of a rotation. Magenta lines represent an object manipulation.

(c) Spatial proximity between the end effector and objects. Black arrows represent an object manipulation, black crosses a previous detected task-critical event. The candidate shown as a blue circle is not considered task-critical event because the distance to an already identified events is too small (parameter min_segment_duration). The red circle is not considered task-critical because during this period the object is being manipulated (black dashed line enclosed by arrows).

Figure 4.7: Task-critical events of the pouring task.
Chapter 5

Conclusion & Outlook

5.1 Conclusion

The presented LfD method is able to learn complex object manipulation tasks using Dynamic Movement Primitives. Robot-independent skill encoding, makes the learned skills reusable on different robotic arms, which was proven by the application of these methods using different robots for teaching and reproducing in the experiments. A simplified demonstration interface for the teacher, which only requires a single demonstration of the task is clearly an advantage over other LfD methods that typically require more than one demonstration. The sensor data based segmentation method divides complex tasks into simpler goal-directed subtasks. Thus, the online movement adaption capabilities of DMPs can be applied to object manipulation tasks. Segmentation of movements using task-critical events allows the learning of complex tasks with a single demonstration. However, the segmentation requires an accurate object recognition and is designed to work with object manipulation tasks only.

The the actual movements are encoded on a low-level using DMPs which generate position, velocity and acceleration values. Tasks are segmented in subtasks where each segment is considered as unique. Therefore, the segmentation method is not semantic which means that a segment has no meaning. Thus, a subtask can define an arbitrary movement, not only a set of known movements.

5.2 Outlook

Due to the lack of available hardware, the results of the LfD approach were only tested in simulation. Future work will address this problem and provide an implementation of a DMP controller for a real robot arm. To make this possible, the robot arm must provide
the ability to control each joint individually with a high update rate.

Regression Method

While the representation of the function $f$ (see Equation 2.3) with a fixed number of learning parameters (weights $w_i$) works well, approximation errors are possible. The presented segmentation method creates subtasks from task features and does not take the approximation error into account. Therefore, it is possible that a movement of a subtask is not a good imitation of the demonstrated movement. An improved version of the approximation method should choose the number of weights and their properties (center and height), depending on the shape of the curve, automatically. The Receptive Field Weighted Regression (RFWR) looks like it could provide such an improvement ([SA98]).

Another option might be the approximation of $f$ with a more sophisticated supervised learning method, for example Gaussian Processes (GP) ([RW06]). The duration of a movement and the frequency of the captured data defines the number of data points in the training set. The training cost for a GP has $O(N^3)$ complexity, where $N$ is the number of training samples. This is caused by an inversion of the $N \times N$ covariance matrix. Even relatively short movements of 10 seconds with a data rate of 1000 Hz result in 10,000 data points which leads to a learning time of $C \times 10^9$, where $C$ is the computation constant. A method called Sparse Pseudo-input Gaussian Processes reduces the complexity of the training cost to $O(NM^2)$, where $M$ is a user chosen number much smaller than $N$ ([Snc07]). Early tests of these method resulted in a bad approximation of $f$. But there are many parameters for fine-tuning which need further examination.

Segmentation as Supervised Learning Problem

Another interesting improvement would be the learning of segmentation criteria. Instead of fixed rules applied to a set of sensor data, the teacher provides segmentation points which serve as target for a classification problem. Segmentation points are only recorded in an initial setup mode. The classification problem can be described as follows, for every state of the movement there is a decision if the movement should be segmented at this position or not. New sensor data could be easily added as input features with such an approach. But, the selection of input features for the classification needs further study.

High-Level LfD with DMPs

Generating movements using DMPs for high-level LfD approaches (as in [CE12]) would be an interesting combination of high-level and low-level LfD. Such a combination enables abstract concepts and the required movements of the action to be learned. The advantage of a high level conceptual representation is that the learned tasks can be more abstract such
as: “stack all red cubes”. The actual movements are typically generated by a trajectory planner. In contrast to DMPs, trajectory planners are not able to adapt a movement online to changes in the environment. Further, a movement plan generated by a trajectory planner does not reproduce a specific shape of a movement. DMPs allows the teacher to influence the shape of the movement which in turn enables the usage for tasks that require both, abstract task encoding and sophisticated motor skills.
Appendix A

Locally Weighted Regression

The general purpose of supervised learning algorithms is to find a function which properly approximates a given set of training examples $D = \{(x_1, y_1), ..., (x_n, y_n)\}$. For a continuous function this process is also known as regression analysis. In most learning methods, a single global parametrized model is used to approximate the training data. This model serves to generate predictions while the training data is discarded. Lazy learning methods predict the value using the training data at the time of a query. As during the learning process only the training data is saved, such methods are also known as memory-based learning.

If the training data is available at query time it is possible to fit a local model to points around the query point $q$ which serves as an estimate for the result $\hat{y}$. The nearest neighbor algorithm is a simple method which chooses the closest point $x_i$ according to $d(q, x_i) \leq d(q, x_j), \forall i \neq j$ where $d(q, x)$ denotes a distance function (typically the Euclidean distance). The prediction is given by the corresponding value to $x_j$, $\hat{y} := y_j$. An extended version of this averages over the $k$ closest points is known as $k$-nearest neighbor algorithm.

In Locally Weighted Regression (LWR), as local models, global structures like linear functions or polynomial functions are used. These global structures are fitted according to a locally weighted training method. The fitting corresponds to the problem of finding the parameters $\beta_i$ of the global models. Once using global linear models this can be expressed as

$$x_i^T \beta = y_i,$$

with the vectors $x_i \in \mathbb{R}^{k+1}$, $x_i = (1, x_{i1}, ..., x_{ik})^T$ and the parameters $\beta \in \mathbb{R}^{k+1}$, $\beta = (\beta_0, ..., \beta_k)^T$ or in matrix form as

$$X \beta = y,$$

with the vectors $x_i$ as rows of $X \in \mathbb{R}^{n \times (k+1)}$

$$X = \begin{bmatrix} x^T_1 \\ \vdots \\ x^T_v \end{bmatrix}.$$
Estimation of the parameters $\beta$ can be done with the least squares method which minimize the unweighted error criterion

$$E = \sum_i (x_i^T \beta - y_i)^2$$

solving the normal equations for $\beta$ leads to

$$\hat{\beta} = (X^TX)^{-1}X^Ty.$$ 

The resulting $\beta$ tries to fit all points in the data set by a linear function. In order to turn the global model into a local model, the distance between query point $q$ and each point $x_i$ is taken into account:

$$E(q) = \sum_i (x_i^T \beta - y_i)^2 \psi_i(q)$$

where $\psi$ denotes a kernel function, for example a squared exponential function $\psi_i(q) = exp(-d(x_i,q)^2)$. This function weights each data point by the distance between the query point $q$ and $x_i$. If the distance is small the value of $\psi$ is close to 1 which results in a higher impact on the output values. A greater distance of the points leads to a weight close to 0 which minimizes the effect of far distant points. The parameters $\beta$ which minimizes the error criterion $E(q)$ are given by

$$\hat{\beta} = (X^T WX)^{-1}X^TWy,$$  \hspace{1cm} (A.1)

where $W \in \mathbb{R}^{n \times n}$ is a diagonal matrix with elements $W_{ii} = \psi_i(q)$. A prediction of $\hat{y}(q)$ is calculated by

$$\hat{y}(q) = q^T \hat{\beta} = q^T(X^T WX)^{-1}X^T W y$$

Note, many different distance functions $d$ as well as kernel functions $\psi$ can be used with this method.

As the prediction of a value involves a lot of computational effort, LWR is expensive. Further, the required computational time depends on the size of the data set. Therefore, LWR is not suited for robot control with real time requirements. However, with some assumptions the lazy learning method can be turned to an eager learning method. This is achieved by computing the regression at the learning time, before a prediction was requested. Thus, using the query point $q$ to define the center of the kernel function $\psi$ is not possible anymore. Instead of a query specific $\psi$ function, a predefined number of kernel functions $\psi_l$, with $l = 1...r$ are defined. In case of squared exponential kernel functions $\psi_l(x) = exp(-h_l(x - c_i)^2)$, the center $c_i$ and the bandwidth $h_i$ remain parameters which are
chosen as required by the learning task (e.g. equally spaced kernels). The parameters \( \hat{\beta}^i \) of the linear models are estimated as defined in Equation [A.1]. For prediction each parameter \( \hat{\beta}^i \) is weighted with the kernel functions \( \psi_i \). Therefore prediction of a query point \( q \) is given by

\[
\hat{y}(q) = \frac{\sum_i \psi_i(q)q^T \hat{\beta}^i}{\sum_i \psi_i(q)}.
\]

An example with 10 kernel functions is shown in Figure [A.1].

If \( x \) and \( y \) are scalar values and the offset of the linear model (constant 1 in \( x_i \)) is omitted the Equation [A.1] is reduced to

\[
\hat{\beta}^i = \sum_{i=1}^{n} \frac{x_i y_i \psi_i(x_i)}{x_i^2 \psi_i(x_i)}.
\]

With the prediction of \( \hat{y} \) as

\[
\hat{y}(q) = \frac{\sum_i \psi_i(q)q^T \hat{\beta}^i}{\sum_i \psi_i(q)}.
\]

A complete survey of Locally Weighted Learning (including LWR) can be found in [AMS97].
Figure A.1: LWR Example. (1) Training samples (generated by the function $g(x) = \sin(10x)x + \epsilon$ with random normal distributed noise $\epsilon$) and predicted function. Local linear models (2) for the $|w| = 10$ equally spaced weight functions (3).
Hierarchical Clustering

Clustering is an unsupervised learning problem, as any of these learning problems its purpose is to find a structure in the data. A cluster can be described as a group of elements which share a certain similarity. This similarity is often defined by spatial proximity which can be measured by a distance function.

Hierarchical Clustering (Joh67) starts by creating a cluster for each of the \( N \) elements in the dataset. Each of the \( N \) clusters now contains exactly one element. The two most similar clusters are united into one cluster. This process of uniting clusters is repeated until there is only one cluster with \( N \) elements left. Alternative ending conditions are: A defined number of clusters is reached, the distance between each cluster is greater than a threshold.

The similarity of two clusters can be measured in many different ways, three different kinds are common: single-linkage, complete-linkage and average-linkage clustering. In single-linkage clustering the distance between two clusters \( A, B \) is equal to the shortest distance between any element of cluster \( A \) and any element of cluster \( B \). In complete-linkage instead of the shortest distance, the greatest distance of the elements is used. In average-linkage the average distance of the elements defines the distance between two clusters.

A simple version of the single-linkage algorithm is defined as follows:
Algorithm B.1 Single-Linkage Clustering [Mat12]

The dataset contains $N$ elements. The $N \times N$ distance matrix $D$ is initialized as $D_{i,j} = d(i, j)$ where $d$ is any distance function for two elements. $L(k)$ is the level of the $k$th clustering. $m$ denotes the sequence number. Parameter: $D_{\text{min}}$

1. Begin with Level $L(0) = 0$ and sequence $m = 0$

2. Find the most similar pair of clusters $r, s$ in the distance matrix

   $d(r, s) = \min d(i, j)$

3. Increment the sequence number: $m = m + 1$. Merge clusters $r, s$ into a single cluster. Set the level of this clustering to:

   $L(m) = d(r, s)$

4. Update the distance matrix $D$ by deleting the rows and columns of cluster $r$ and $s$. Add a row and column of the distance to the newly formed cluster. The distance between the new cluster $(r, s)$ and an old cluster $k$ is given by:

   $d(k, (r, s)) = \min d(k, r), d(k, s)$

5. If $\min d(i, j) \leq D_{\text{min}}$, stop. Else, go to step 2.

The complexity of the algorithm B.1 is $\Theta(n^3)$, but an optimal version with $\Theta(n^2)$ is available as SLINK in [Sib73].
Figure B.1: Example of Hierarchical Clustering with 40 random data points between 0 and 1. The center of the figure is a colored map of the $40 \times 40$ distance matrix $D$. Two different clustering methods are shown as dendrograms, single linkage method (top) and complete linkage method (left).
List of Figures

1.1 Generalizing over multiple demonstrations (task space coordinates in mm) \[Sch09\] ......................................................... 4

1.2 3D movement generated by a dynamic system. Green shows the demonstrated trajectory, red is the reproduction with the same start and goal position, and blue represents the adapted movement with a changed goal position. .................................................. 5

2.1 Position, velocity and acceleration of an executed DMP without any external force \(f = 0, K = 50, D = 12.5, \tau = 1.0, x_0 = 1.1, g = 2.2\) ...................................................... 8

2.2 Canonical system \(s\) with different values of \(\tau\) ...................................................... 9

2.3 (1) LRWR learned target function \(f_{\text{predicted}}\) (green dashed) and the actual \(f_{\text{target}}\) (blue); (2) 8 Gaussian weight functions, \(|w| = 8\); (3) Local linear models .................................................. 10

2.4 Position, velocity and acceleration of a minimum jerk trajectory (start: 0.3, goal: 1.4) (blue); DMP movement with learned function \(f\) plotted in Figure 2.3 setup with initial start and goal (green dashed curve). ...................................................... 10

2.5 Sketch of a one-dimensional DMP \([PHAS09]\) ...................................................... 11

2.6 Adaption of a DMP to a new start (+0.4) and goal (+0.2) position. ...................................................... 11

2.7 Sketch of a multi dimensional DMP ...................................................... 12

2.8 Drawbacks of the original DMP formulation. In the left Figure the sign of \((g - x_0)\) has changed and the movement is mirrored (red line). The changes of the goal in the right Figure are extremely small (-0.02), but the movement between start and goal position doesn’t adapt as expected. ...................................................... 13

2.9 Adaption to new goals with the modified DMP version. Compared to Figure 2.8 this is a huge improvement in adaption accuracy. ...................................................... 13

3.1 Differently shaped grippers ...................................................... 18

3.2 Sketch of the 4-dimensional robot-independent DMP. Three transformation systems for the position of the end effector in 3D Cartesian task space and one transformation system for the orientation of the end effector (using special quaternion formulation from \[PRKS11]\) ...................................................... 19

3.3 Simplified input/output view on the learning phase ...................................................... 20
LIST OF FIGURES

3.4 Velocity of the gripper fingers (blue), the green line is the Gaussian-smoothed version of the velocity signal. The ignored area (noise prone) is marked with an upper and a lower margin illustrated as gray horizontal lines. The blue areas indicate the object manipulation units found by clustering of all points lying outside of the ignored area. Red circles indicate a gripper release, cyan squares a grasp. 23

3.5 Velocity of the roll angle (blue), the green line is the Gaussian-smoothed version of the velocity signal. The ignored area (noise prone) is marked with an upper and a lower margin illustrated as gray horizontal lines. The blue areas indicate the event units found by clustering of all points lying outside of the ignored area. Red diamonds indicate actual event time. Times in which an object is manipulated is marked by a magenta line with arrows at both ends. 24

3.6 3D task space trajectory segmented into two parts (blue and green), the arrows indicate the distance of the task-critical event (red diamond) to the objects (square points). Obj1 is chosen as reference object because of the smaller distance to the end effector at the time of event. 25

3.7 Euclidean distance between objects and the position of the end effector. The black crosses represent previously identified task-critical events. Manipulation of an object is marked with a black dashed line. A valid task-critical event is marked with a green circle. Blue and red circles are rejected event candidates – blue means the minimum segment length would be violated by splitting at this time – red indicates that the object is manipulated at this time. 26

3.8 Example of a segmented trajectory: (a) approach obj1, (b) grasp obj1, (c) move obj1 to obj2 and release obj1, (d) move to home position. Each red diamond represents a task-critical event labeled with a number indicating the reference object. The gripper action is written in square brackets (release/grasp). 27

4.1 KATE 31

4.2 Coke in cup task – reproduction in Gazebo with PR2 Robot. 33

4.3 3D task space trajectories of the coke in cup task. 33

4.4 Task-critical events of the “coke in cup” task. 34

4.5 Pouring task - reproduction in Gazebo with PR2 Robot. 35

4.6 3D task space trajectories of the pouring task. 35

4.7 Task-critical events of the pouring task. 36

A.1 LWR Example. (1) Training samples (generated by the function \( g(x) = \sin(10x)x + \epsilon \) with random normal distributed noise \( \epsilon \)) and predicted function. Local linear models (2) for the \( |w| = 10 \) equally spaced weight functions (3). 44
| B.1 | Example of Hierarchical Clustering with 40 random data points between 0 and 1. The center of the figure is a colored map of the $40 \times 40$ distance matrix $D$. Two different clustering methods are shown as dendrograms, single linkage method (top) and complete linkage method (left). | . . . . . | 47 |
Bibliography


2.1


2.1


3.2


1.2.4


4.1


5.2


5.2


1.2.4


1.1, 1.2.4

