Learning from Demonstration by Averaging Trajectories

Project at the
Zentrum für angewandte Forschung an Hochschulen
Robotic Laboratory of the
University of Ravensburg-Weingarten

Supervisor: Prof. Dr. Wolfgang Ertel

Heiko Posenauer

March 26, 2012

Contents

Chapter 1

Introduction

Machine Learning is a branch of Artificial Intelligence. The goal is to enable robots to learn tasks in contrast to 'hard code' behaviour patterns like the code of practice in strict deterministic environments like manufacturing (industrial robots).

The different approaches to achieve this goal can be split into two categories:

1. Learning though experience (by means of sensors)
2. Teacher based learning (a human or a robot giving input)

Learning from Demonstration (LfD) is a teacher based method (as the name implies), whereby the robot learns a complex task by one or more demonstrations. Therefore the demonstrations need to be recorded and translated into a language the robot understands, then the information of interest can be extracted and used as a model when the robot performs the task on its own.

The goal of this project is to introduce and implement a method to produce this model and an algorithm to reproduce the task on the model's basis.

The context of this method has been described in chapter 2. Chapter 3 covers the mathematical background of the algorithm and in chapter 4 the usage of the program Leatra (Learning Trajectories) that implements the algorithm is depicted.

Chapter 2

Learning from Demonstration by Averaging Trajectories

The "Learning from Demonstration by Averaging Trajectories" (LAT) method supports the learning of tasks with involved objects. This means that at least one object and its position needs to be involved, serving as a point of reference. There are three phases the robot needs to traverse in order to learn a task:

1. Demonstration
2. Generalisation
3. Reproduction / Anticipation

The trajectories and object coordinates for a certain task are gained in the demonstration phase. The generalisation phase reduces this data to a model, that provides information on how to behave regarding the objects involved. Finally, in the third - reproduction phase, the learning algorithm is applied. It finds a solution for the task on the basis of the model and the current situation in the robots environment (found objects and their coordinates).

If not noted differently, the horizontal axis of all graphs in this chapter is the time axis.

2.1 Demonstration

Before being able to perform a certain task, the robot needs to learn it through demonstration. For the LAT method objects need to be involved, which the robot recognises. Once the initial coordinates of the objects are known, during the demonstration the trajectories relative to the objects can be recorded. In

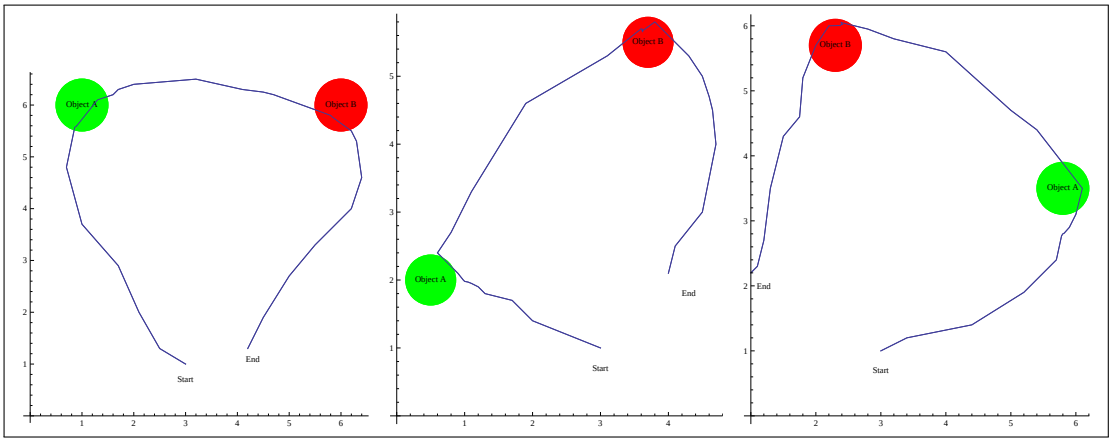

Figure 2.1: Fictional demonstration 1 to 3 (horizontal axis: x, vertical axis: y), with object A (green) and B (red)

this way the robot gains a trajectory (in n dimensions) for each object - each trajectory containing the distance between object and robot end effector. The demonstration of the task should be repeated several (k) times, before it can be moved on to the next phase.

In figure 2.1 (page 4) three fictional trajectories, and the two objects A and B are displayed. These trajectories can be gained for instance through a demonstration by a human, leading the robot arm. The measured coordinates are for example the end effector position. As it easily can be seen from the three diagrams, object A is approached first, followed by object B. The objects take different starting positions during the three demonstrations. Those starting positions are recorded at the beginning, before the recording of the trajectory itself. Any changes of the object's coordinates during the demonstration are disregarded.

Looking only at the distance of the robot effector to an object, e.g. the x coordinates and object A, the curves would look like in figure 2.2 (page 5).

At the end of this phase there will be k recorded trajectories (each made out of many dimensions) for each object involved.

2.2 Generalisation

Having the k recorded demonstrations, the data of the trajectories (like in figure 2.2) is reduced to its mean and its standard deviation.

Calculating the mean over all the trajectories from figure 2.2 (page 5), one gets a graph like in figure 2.3 (page 5), in which the brighter area around describes the standard deviation. This can be interpreted as a characteristic behaviour regarding the x-axis and object A.

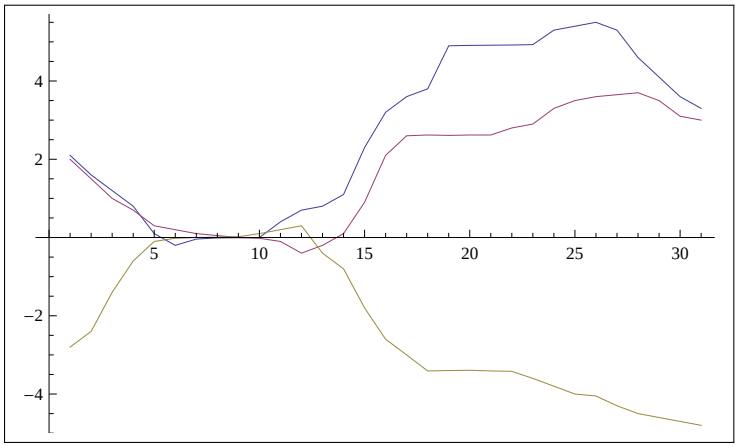

Figure 2.2: Distance between the robot's end effector and object A on the x axis of all three demonstrations. blue: demonstration 1, red: demonstration 2, brown: demonstration 3 (see figure 2.1)

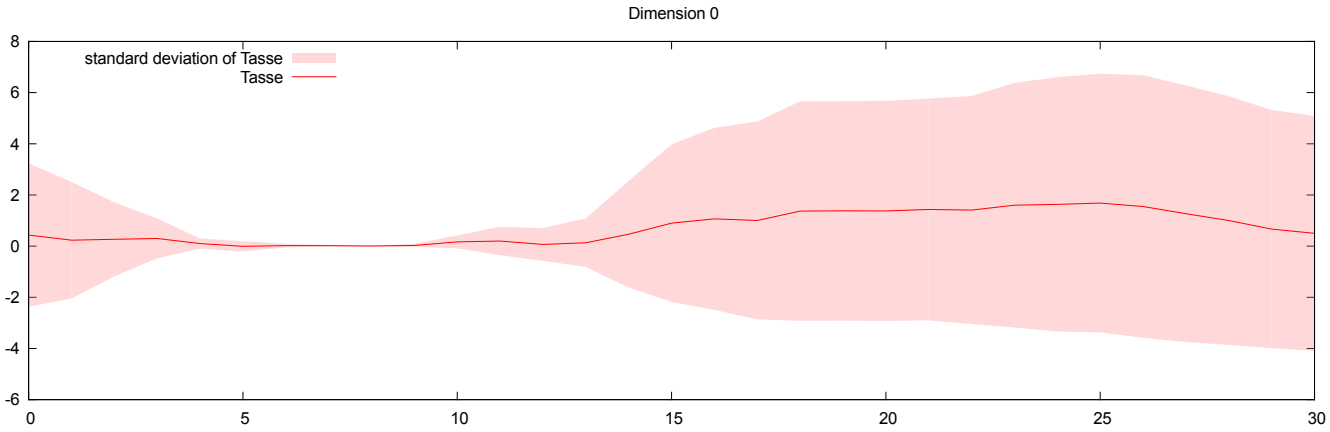

Figure 2.3: Mean of distances to A on the x-axis, surrounding curve is the standard deviation; In this work, this is called 'model of A' (see figure 2.2)

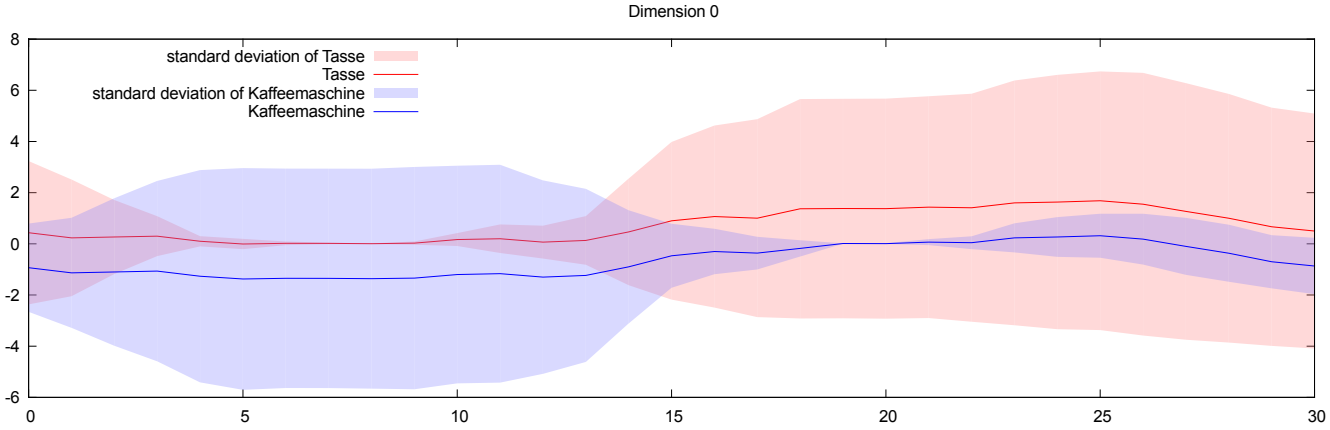

Figure 2.4: Model for the two objects A and B regarding the x-axis (see figure 2.3)

This way a characteristic motion regarding all objects and axes can be found and some noise will be flattened. Now the robot has a model, on how to behave for this task regarding the different objects (see figure 2.4, page 6 and figure 2.5, page 7). The data is stored, together with the detected objects and used again, when the task has to be performed. (Alternatively this phase (calculation of the mean and standard deviation) also could be integrated in phase 3, making the method less eager.)

2.3 Reproduction / Anticipation

When the robot has to perform the task on its own, first it needs to recognise the objects and their coordinates. These coordinates serve as an offset for the averaged trajectories from phase 2. See figure 2.6 (page 7), the two curves describing the characteristic behaviour regarding A and B in x-axis, have been shifted to the current locations of the objects detected (y-axis: figure 2.7, page 8). Having thus all models of all involved objects in the right coordinate system as a second step, the robot has to find one trajectory that corresponds to all. This is achieved by averaging the trajectories.

The implementation of this algorithm (described below) in the program Leatra was the main task of this project. In chapter 4 it is described, on how to use Leatra.

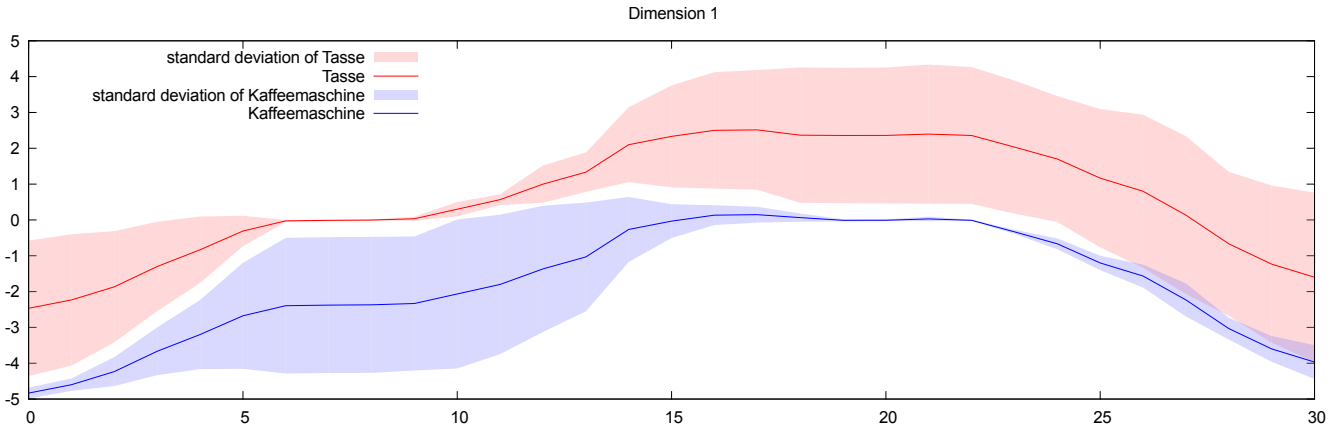

Figure 2.5: Model for the two objects A and B regarding the y-axis

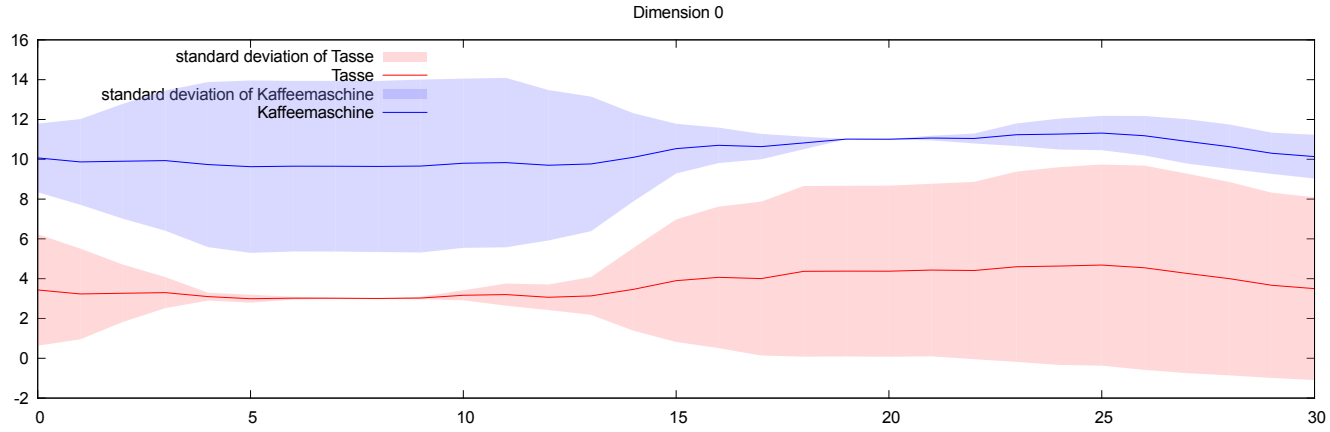

Figure 2.6: Model of the x-axis has been shifted to the current position of the objects (compare to figure 2.4)

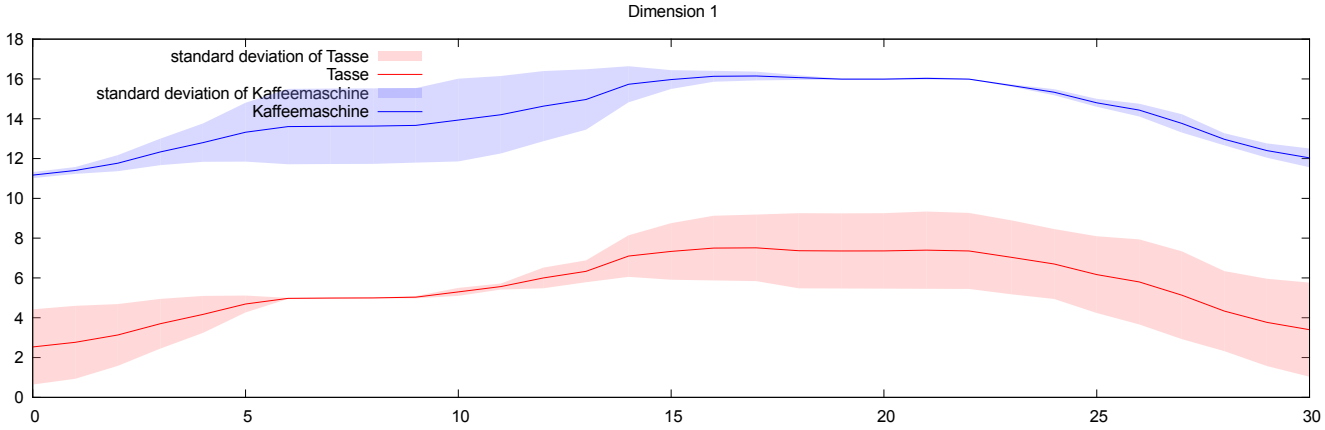

Figure 2.7: Model of the y-axis has been shifted to the current position of the objects (compare to figure 2.5)

2.3.1 Averaging trajectories

Each object has its own trajectory, plus the standard deviation at each point (figure 2.5 and 2.5, page 7). A small standard deviation points to the fact, that in the demonstration sessions this point (distance to a certain object) had to be strictly observed. In turn a big standard deviation points out, that this point (distance to a certain object) was often very different and is less a constraint in the trajectory. Having the standard deviation and the mean, the normal distribution is known for each recorded point in time.

By calculating the product over all normal distributions (see chapter 3), the resulting distribution has its mean at the point, where the strongest constraint attracts it to. This point is the new mean for the specific point of the new trajectory. The sequence of all means calculated this way make up the new trajectory.

As in figure 2.9 and 2.8 (page 9), can be seen, the learned (averaged) trajectory in green satisfies the constraints of both models.

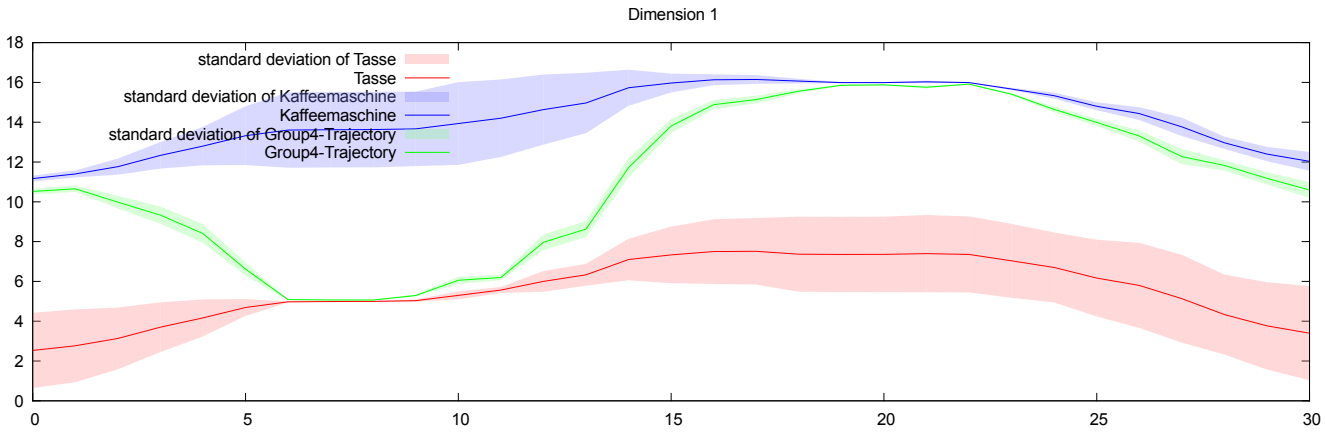

Figure 2.8: The green graph is the resulting trajectory (y-axis).

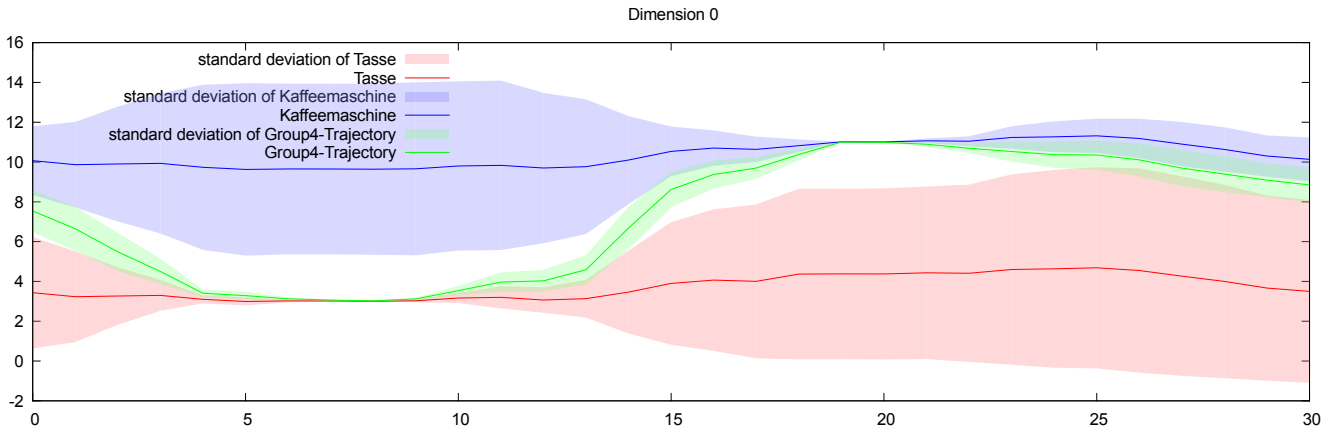

Figure 2.9: The green graph is the resulting trajectory (x-axis).

Chapter 3

Mathematical Background

The standard form of the Probability Density Function (PDF):

$$f_{\sigma,\mu}(t) = \frac{1}{\sigma\sqrt{2 * \pi}} e^{-\frac{(t-\mu)^2}{2\sigma^2}} \tag{3.1}$$

The product of multiple PDFs is not automatically a PDF in turn, but it is proportional to another PDF (Figure 3.1, page 10). The integral (the area under the curve) of the product is not equal to one. But scaled with a factor it can also meet this requirement.

A derivation of a proportional PDF for the kind of green curves in figure 3.1 is

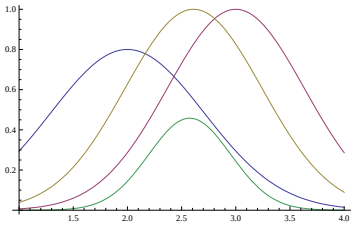

Figure 3.1: The green curve is the product of the other curves (all PDFs)

universally - for the product of n PDFs - specified below. Some inspiration could be gained in the chapter "Gaussian Function Properties" of the book "Spectral Audio Signal Processing" by Smith [1].

The benefit of determining a proportional PDF instead of calculating the maximum of the product of the PDFs is its simplicity and the fact, that the result can be handled and interpreted as a Gaussian distribution.

Having n standard Gaussian PDFs with different σ and μ:

$$f_{\sigma_n,\mu_n}(t) = \frac{1}{\sigma_n\sqrt{2*\pi}} e^{-\frac{(t-\mu_n)^2}{2\sigma_n^2}} \tag{3.2}$$

The product over $f_{\sigma_1,\mu_1}(t)$ to $f_{\sigma_n,\mu_n}(t)$:

$$\prod_{i=1}^{n} f_{\sigma_i,\mu_i}(t) = \prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2*\pi}} e^{-\frac{(t-\mu_i)^2}{2\sigma_i^2}} \tag{3.3}$$

Separate the product:

$$= \left(\prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2*\pi}}\right) e^{-\sum_{i=1}^{n}\frac{(t-\mu_i)^2}{2\sigma_i^2}} \tag{3.4}$$

Expand the numerator in the exponent:

$$= \left(\prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2*\pi}}\right) e^{-\sum_{i=1}^{n}\frac{t^2-2t\mu_i+\mu_i^2}{2\sigma_i^2}} \tag{3.5}$$

In order to bring the n exponents on the same denominator, expand fraction with $(\frac{1}{\sigma_i^2})$:

$$= \left(\prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2*\pi}}\right) e^{-\sum_{i=1}^{n}\frac{t^2(\frac{1}{\sigma_i^2})-2t\mu_i(\frac{1}{\sigma_i^2})+\mu_i^2(\frac{1}{\sigma_i^2})}{2}} \tag{3.6}$$

Dragging the sum into the numerator:

$$= \left(\prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2*\pi}}\right) e^{-\frac{t^2\left[\sum_{i=1}^{n}(\frac{1}{\sigma_i^2})\right]-2t\left[\sum_{i=1}^{n}\mu_i(\frac{1}{\sigma_i^2})\right]+\left[\sum_{i=1}^{n}\mu_i^2(\frac{1}{\sigma_i^2})\right]}{2}} \tag{3.7}$$

Expand fraction with $\left[\sum_{i=1}^{n}\frac{1}{\sigma_i^2}\right]^{-1}$:

$$= \left(\prod_{i=1}^{n} \frac{1}{\sigma_i\sqrt{2*\pi}}\right) e^{-\frac{t^2-2t\frac{\sum_{i=1}^{n}\mu_i\frac{1}{\sigma_i^2}}{\sum_{i=1}^{n}\frac{1}{\sigma_i^2}}+\frac{\sum_{i=1}^{n}\mu_i^2\frac{1}{\sigma_i^2}}{\sum_{i=1}^{n}\frac{1}{\sigma_i^2}}}{2\frac{1}{\sum_{i=1}^{n}\frac{1}{\sigma_i^2}}}} \tag{3.8}$$

Equating the coefficients by comparing to the Gaussian standard form $\rightarrow$ see equation (3.1):

$$\mu = \frac{\sum_{i=1}^{n} \mu_i \frac{1}{\sigma_i^2}}{\sum_{i=1}^{n} \frac{1}{\sigma_i^2}} \tag{3.9}$$

$$\sigma^2 = \frac{1}{\sum_{i=1}^{n} \frac{1}{\sigma_i^2}} \tag{3.10}$$

These two equations (3.9) and (3.10) provide all information, in order to conclude to the resulting PDF. They are used in Leatra (chapter 4) in order to calculate the resulting trajectory.

Chapter 4

LEATRA

Leatra is a module in order to perform the kind of function approximation described in the chapter above. Leatra stands for Learning Trajectories.

This chapter provides information for the installation of leatra and the usage of the terminal version (see chapter 4.4). Furthermore it explaines the different file types and data types used. In order to work with the source code, the html manual provides extensive information about classes and methods: leatra/src/html/index.html.

4.1 System requirements

Leatra has been tested on Ubuntu 11.10 and MAC OS X, Version 10.7.3. Apart of the g++ compiler in order to install Leatra, the following programs are required:

Gnuplot Linux: `apt-get gnuplot`, Mac: `port install gnuplot`

Latex Linux: `apt-get texlive`, Mac: `port install texlive`

Doxygen only needed for documentation purposes for further extensions of Leatra (Linux: `apt-get install doxygen`, MAC: `port install doxygen`)

4.2 Installation of Leatra

The Leatra main directory needs to be copied to the desired place in the directory tree. On some systems for the installation superuser/root privileges are necessary. In the `leatra/src/` folder a makefile controls the compilation procedure: switch to the src folder and type: make. `leatra/src$ make` Switching back one level higher, to the Leatra folder, now there should exist the executable `leatra` file.

4.3 Leatra File Conventions

4.3.1 The Data File

During the recording of a trajectory data is produced. Leatra can read data from file if it is structured as follows: Each column contains the data of one dimension. The columns need to be separated by a space or a tab. All columns need to be of the same size (The data file is a representation of the Leatra data type ndmap, see also chapter 4.5.1). Example file `demo1` with two dimensions:

```
2.3         9.3
2.5         9.1
2.8         8.6
...         ...
```

4.3.2 The Trajectory File

In order to make the involved objects known to Leatra, the trajectory file is needed. For example, if there have been three demonstrations (demo1,demo2 and demo3) Leatra needs to know all coordinates of the involved objects at all demonstrations. The trajectory file needs do have the ending ".tra" and needs to be structured as follows: Alternating, one line contains the name of the data file (a trajectory), the following line contains the names of the objects, followed in brackets their coordinates. Example file `MakeCoffee.tra`:

```
demo1
Cup(1.0,6.0)Coffee_maker(6.0,6.0)
demo2
Cup(0.5,2.0)Coffee_maker(3.7,5.5)
demo3
Cup(5.8,3.5)Coffee_maker(2.3,7.5)
```

Note that there are no blank allowed anywhere! This file needs to be placed in a sub folder in the 'leatra' directory carrying the same name like the trajectory file (including the ending .tra), together with the data files:

```
leatra/MakeCoffee.tra/$ ls
MakeCoffee.tra  demo1  demo2  demo3
```

All Trajectory Files (here: demo1,demo2 and demo3) need to have the same number of recorded points and the same number of dimensions. The '.tra' file

must not contain any underscore characters.

4.3.3 The Model File

The model file is produced automatically by Leatra, when saving a model to file. As a model consists out of information regarding several objects, again several trajectories are needed. The model for one object consists out of the following trajectories:

1. mean plus standard deviation (mean+stdev)
2. mean
3. mean minus standard deviation (mean-stdev)
4. standard deviation (stdev)

The order of the files needs to be observed. Example file `Coffee.mod`:

```
Cup
Cup_mean+stdev;Cup_mean;Cup_mean-stdev;Cup_stdev;
Coffee_machine
CM_mean+stdev;CM_mean;CM_mean-stdev;CM_stdev;
```

The model "Coffee.mod" carries models for two objects: the Cup and the Coffee_machine. For each object there are the four saved trajectories mentioned above, separated by a semicolon. The names used here (Cup_mean+stdev ...) are the names of the 'Data Files' saved in the same directory.

The Model file needs to be placed in a sub folder in the 'leatra' directory with the same name like the model file (including the ending .mod), together with the data files:

```
leatra/Coffee.mod/$ ls
Coffee.mod  Cup_mean+stdev  Cup_mean  Cup_mean-stdev  Cup_stdev
CM_mean+stdev  CM_mean  CM_mean-stdev  CM_stdev
```

The '.mod' file must not contain any underscore characters.

4.3.4 The Object File

Through the object file Leatra can read information about the position of several objects. It consists of only one line, enumerating the objects, followed in brackets

by coordinates. Please note, there are no spaces used anywhere. Example object file `objects`:

```
Cup(0.0,2.1)Coffee_maker(3.1,12.8)
```

The object file, needs to be located in the 'leatra' directory, or the input for the object file name needs to be completed by the path.

4.4 Leatra Terminal Version

The command line version provides a user friendly interface for a limited number of Leatra functions. By typing `./leatra -h` a command-line help appears, giving instructions on how to use it.

Usage: `./leatra [-dt [optionally -sm] or -rm] -r [-p or -st]`

Meaning of the keys:

-dt "demonstration trajectories": read trajectory data from file - must be followed by the name of the sub folder (see chapter 4.3.2)

-rm "read model": read a model from file, must be followed by the name of the sub folder (see chapter 4.3.3)

-sm "save model": save the produced model to a file (will create a file structure like defined in chapter 4.3.3)

-r "reproduction": compute a trajectory for the before defined model. This parameter needs to be followed by the name of an object file (see chapter 4.3.4)

-p "print": plots a pdf of the results to the 'leatra' directory

-st "save trajectory": saves the calculated trajectory to file as a map (see chapter 4.5.1).

Example usage

In the leatra directory there is a set of saved example trajectories (the source of figure 2.1 on page 4), in the file MkCoffee.tra. In order to load this trajectories to leatra and to save the calculated model to file type:

```
./leatra -dt MkCoffee -sm
```

Now a model exists. It is saved in the folder 'MkCoffee.mod' and can be loaded by:

```
./leatra -rm MkCoffee
```

Once Leatra has a model (either from Model File or from Trajectory File) it can be applied to a specific situation, defined by an Object File. There exists an example Object File in the leatra directory: objects.txt.

```
./leatra -rt MkCoffee -r objects.txt
```

The command above creates a new trajectory on the basis of the model, produced through the Trajectory File and the Object File. In order to safe the new trajectory to file and to plot the resulting trajectory type:

```
./leatra -rt MkCoffee -r objects.txt -st -p
```

As a result a pdf file (MkCoffee_apx.pdf), illustrating the approximated trajectory (like in figure 2.9 and 2.9 on page 9) and a folder named 'MkCoffee-Trajectory_mean', carrying the calculated trajectory in a Data File has been created.

4.5 Leatra Data structures

Leatra uses three different data structures:

ndmap n-dimensional map

ndmapSet a list or set of ndmaps

ndmapSetGroup a list or set of ndmapSets

During the recording of a trajectory the data points of all dimensions can be saved in one ndmap. Ndmaps can be grouped in a ndmapSet and ndmapSets can be grouped in a ndmapSetGroup.

4.5.1 ndmap Data Type

The ndmap (map) can be described as a two dimensional matrix. The depth of the matrix is determined by the number of values recorded and the width by the

amount of dimensions. The number of values in each column (time axis) has to be constant. The size of the intervals needs to be constant as well (on a time axis: the sampling rate needs to be exact periodic). The smallest ndmap possible is a 1 by n matrix – a vector. But it can also be extended to a n by m matrix.

4.5.2 ndmapSet Data Type

The ndmapSet (set) is a set or list of ndmaps. The smallest ndmapSet contains one ndmap. Again this data type can carry as many ndmaps as the user wishes.

4.5.3 ndmapSetGroup Data Type

The ndmapSetGroup (group) data type is a list of ndmapSets. The smallest group can carry as many ndmapSets as needed.

4.6 Use of Leatra Classes

This is a very brief description of the use of the classes in Leatra. Further information can be found in the the HTTP manual pages found here: leatra/src/html/index.html.

4.6.1 Class trajectory

The class trajectory represents one demonstration (it contains one ndmap and a list of involved objects).

4.6.2 Class object

The class object represents an object - defined by its name and its coordinates.

4.6.3 Class approximation

The class approximation provides all tools for the learning algorithms.

Method make_model

The method make_model takes as parameter a list (standard deque) of trajectories and returns a model in form of a ndmapSetGroup. The ndmapSetGroup provides the method 'add_offset', taking as a parameter a list (deque) of objects and uses them as an offset for the model (see chapter 2.3 and figure 2.6).

Method constraint_fusion

The method constraint_fusion takes as parameter a ndmapSetGroup - having the form of a model (the offset needs to be added already). This method applies the algorithm discussed in chapter 2.3.1 and the mathematical base in chapter 3. The return value is a set of three ndmaps, carrying the mean (the resulting trajectory) and the mean +/- standard deviation.

Bibliography

[1] Julius O. Smith. Spectral Audio Signal Processing. http://ccrma.stanford.edu/~jos/sasp/, 2011 (accessed March 03., 2012). online book.

List of Figures