Institut für Parallele und Verteilte Systeme
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit Nr. 2350

Analysis of Methods for Segmentation and Representation of Time Series for the Recognition of Motion Pattern

Felix Brucker

Course of Study:	Technische Kybernetik
Examiner:	Prof. Dr. rer. nat. habil. Paul Levi
Supervisor:	Dipl.-Inf. Kai Häussermann
Commenced:	22.08.2011
Completed:	30.03.2012
CR-Classification:	I.2.1, I.2.4, I.2.6, I.2.11, I.5.1, I.5.3

Abstract

In the context of the EU project *RoboEarth* robots shall exchange their knowledge, in the form of platform independent information, in a worldwide network. The goal is that robots get access to knowledge and experience other robots collected to solve so far unknown problems. To provide complex motion sequences in order to solve such problems, it is necessary to represent them in a platform independent way. Therefore, in a pre-process, the movement records have to be divided into different segments and labelled clearly. In this thesis such a technique for performing this task is analysed and tested. Based on multidimensional continuous sensor signals using Microsofts Kinect, motion sequences are divided into single, recurring movement primitives. For this purpose, time series of the positions of different body parts are discretised and represented and later recognised with the help of Hidden Markov Models.

Deutsche Zusammenfassung

Im Rahmen des EU-Projekts RoboEarth soll erlerntes Wissen einzelner Roboter in Form von Plattform-unabhängigen Informationen, innerhalb eines weltweiten Netzwerks, anderen Robotern zur Verfügung gestellt werden. Ziel ist es, dass Roboter für bisher unbekannte Aufgaben vom Wissen und gesammelten Erfahrung anderer Roboter profitieren. Um komplexe Bewegungsabläufe zur Lösung bestimmter Aufgaben anderen Robotern zur Verfügung zu stellen, müssen diese in Plattform-unabhängiger Form repräsentiert werden. Dazu müssen Bewegungsaufzeichnungen zunächst in einer Vorverarbeitung in einzelne Segmente aufgeteilt und anschließend eindeutig bezeichnet werden. In dieser Thesis werden hierfür Bewegungen in Form von Zeitreihen aus mehrdimensionalen Sensorsignalen in zwei Stufen diskretisiert. Bewegungsabläufe werden dabei in kleinere eindeutige Bewegungsprimitive aufgeteilt, welche wiederum mit Hilfe einer Zeitreihenrepräsentation Plattform-unabhängig gemacht werden. Als Sensor dient die Spielkonsole-Erweiterung Kinect von Microsoft. Mit Hilfe von Abstandsberechnungen der Zeitreihenrepräsentationen sowie Hidden Markov Modellen erfolgt anschließend die Erkennung der Bewegungen.

Contents

1 Introduction

1.1 Motivation

So far, most robots were created to manage one specific task. But the problems robots are supposed to solve are becoming more and more complex and therefore robots need to be more versatile. For example, some robots have been created in order to to make them walk or grab objects as best as possible while others have been made to test methods for image, speech or motion recognition. As all the methods and solutions for those problems become better and more stable, one is interested in combining them in a single robot. However it is still very complex and time-consuming to teach a robot a solution for every single problem. An idea to avoid this issue is RoboEarth [1][2]. The goal is that robots worldwide share their knowledge and experience. Each robot which is confronted with an unknown task should be able to check if other robots already solved the problem and then use their experience to accomplish the task. Because every robot is built and equipped differently, it is of course necessary to provide the information in an abstract and universal way.

One kind of information that could be shared through RoboEarth are recognized motion sequences. Again it is necessary to provide the information about those motion sequences in an abstract and platform independent form, so robots with, for example, different kind of sensors are still able to use it. One way to fulfil this condition is to divide these motion sequences into several different parts. The idea is that every motion consists of several specifiable movement primitives, which are transformed into an abstract and platform independent way.

In this thesis this task shall be accomplished with the discretisation and representation of time series. Multidimensional continuous sensor signals are used for time series, which are then discretised and represented in a symbolic domain. These clearly indicated movement primitives can then be recognized and assembled to complete gestures. The idea of the whole procedure is shown in Figure 1.1.

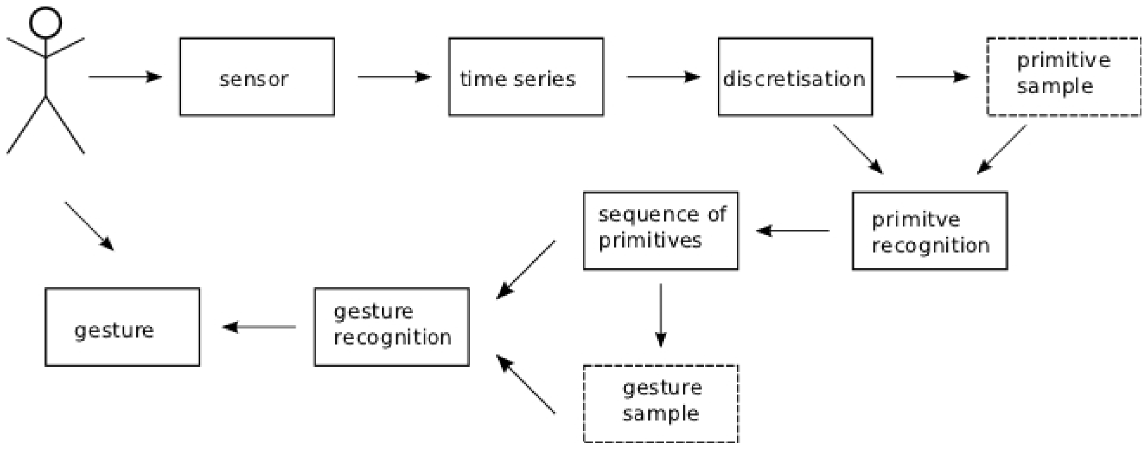

Figure 1.1: *The whole procedure from the movement records to the time series discretisation up to the recognition of the complete gesture. The dashed boxes are the saved platform independent samples, which can be shared between the robots.*

The procedure is split into two parts and provides two kinds of platform independent movement samples. On the one hand are the motion primitives and on the other hand are the complex gestures, which are sequences of the former. Both of them can be shared between robots and used differently. It is possible to let a robot only access the abstract samples of the gestures and define the motion primitives on its own. But it is also possible to access only the samples of the primitives and create new complex gesture out of them.

1.2 Thesis Outline

This thesis shall show, analyse and test a process as described above. The used time series are based on the positions of different body parts in space. These positions are recorded using the Microsoft Kinect system. It is introduced in Chapter 2. Chapter 3 deals with the representation and discretisation of the time series in order to produce the motion primitives. The algorithm which is used for this purpose in this work is called SAX [11][12] and is explained in Section 3.3. Chapter 4 covers the distance measurement, which is necessary for comparing the discretised time series and hence for recognising the motion primitives.

The second part of the described procedure is covered by Chapter 5. It explains Markov Models, especially Hidden Markov Models [14], which are used to recognize the complete gestures out of the sequence of motion primitives. It also handles the Baum-Welch algorithm [13] in Section 5.4, which is used to train the models.

Finally, in Chapter 6 the whole procedure is tested and the results are analysed.

2 The Microsoft Kinect System

There are many different ways to obtain time series. But in most cases they come from some kind of sensor. For gesture recognition in particular, the most used ones are video cameras, accelerator sensors, wired gloves and time-of-flight cameras. This thesis uses the Kinect system from Microsoft shown in Figure 2.1.

Figure 2.1: *The Kinect system from Microsoft.*

The Kinect is an add-on device for Microsofts XBox360, mainly developed by Microsoft itself and PrimeSense [4]. Its purpose is to enable playing video games without any kind of controller by recognising the players movements. While Microsofts competitors Nintendos Wii and Sonys PlayStation Move are using accelerator sensors installed in their controllers, the Kinect mainly uses a depth mapping camera. Altogether, the Kinect consists of a RGB camera, infra-red laser projector, monochrome CMOS sensor, four microphones and a motor, which enables small turns to follow the players.

The most interesting part is of course the IR projector and sensor. The Projector is more precisely a near-infra-red laser [3]. It speckles a grid of dots over the whole room and the players by shining through a micro-patterned plastic lens. The reflected light from all those single spots is recorded by the CMOS sensor. The way the pattern changed reveals the distance of the corresponding part of the grid. With this information a depth map is created as you can see in figure 2.2. The resolution in depth as well as x and y direction is around one centimetre.

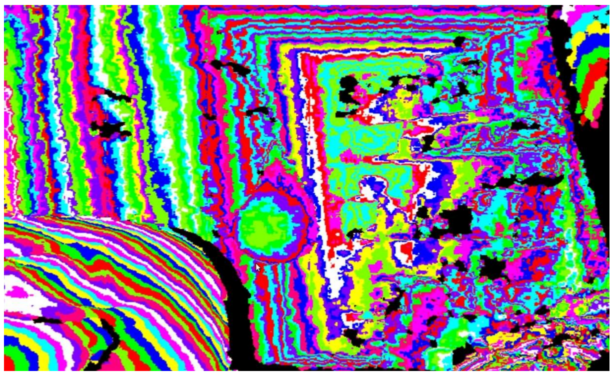

Figure 2.2: *This picture shows a coloured version of the depth map the Kinect is creating.*

Using a near-infra-red laser has the great advantage, that the depth-mapping system is independent of the ambient light situation. Of course, this isn't true for the RGB camera, which is also a CMOS sensor. Both sensors are read out at 30 Hz and provide images with a resolution of 480×640. It is basically used for facial and hand recognition. The microphones enable voice recognition. They are aligned as an array to localise the sound source and to make noise reduction easier. The Kinect can easily be connected via USB.

The software detects every person entering the covered area. By performing a defined pose you get recognized as an active player. According to PrimeSense the software can follow up to 64 persons. Actually, because of computing efficiency, only two persons can be active players. Once a person is recognized as active, a skeleton representing the persons body is calculated, shown in figure 2.3. The skeleton consists of 15 joints like Head, Hands, Knees etc. For each joint the three coordinates are recorded leading to an three dimensional representation of the player. The high resolution of the depth map and noise reduction enable very good results in obtaining time series of these joint positions.

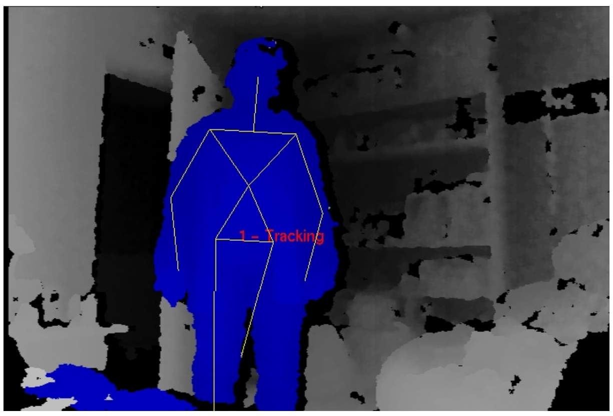

Figure 2.3: *The Kinect detects any person and tracks it. The position of several joints form every active player are calculated resulting in a skeleton.*

The Kinect was first released in the USA on November 4, 2010, followed by Europe and Japan. After 60 days, 8 million units have been sold. According to Guinness World Records its the fastest-selling consumer electronics device, even beating Apples iPhone [5]. On release its price has been around USD 150$ in the USA. Because of this low price and the possibilities its offering, the Kinect is used for many non-playing experiments, often called 'Kinect Hacks', and is quite popular among scientists and engineers (e.g. [3][7][10][8]). At first, different developers created open source drivers. Later, PrimeSense itself published a driver together with its middle-ware 'NITE' which includes the person tracking and skeleton creating functions. In February 2012 Microsoft released a SDK for Windows OS, which also includes the skeleton tracking [6].

This thesis uses the official drivers by PrimeSense as well as an open source C++ wrapper to obtain time series of the different body joints.

3 Representing Time Series

As explained previously in this thesis motion sequences are made platform independent by discretisation and representation of time series. There are many different ways to represent time series. Most of them try to represent the time series as combination of simple functions. The best known are the Discrete Fourier Transformation (DFT), Piecewise Linear Transformation, Discrete Wavelet Transformation (DWT) and Singular Value Decomposition (SVD). Each of those representations has its advantages but also its drawbacks.

Another attempt to represent time series is to use a symbolic strings domain. Their goal is to transform the time series into a string of symbols. A big advantage of these representations is the possibility to extremely reduce the amount of data. Then besides handling just the actual input of the sensor comes the stored data you want to compare with. Comparing time series means in most cases to calculate a distance between them as explained in Chapter 4, or rather between every single point of them. Therefore dealing directly with raw data is very expensive in computing and storing. For this reason one task of working with time series is to reduce the amount of data or the dimensionality without losing its characteristics.

A quite unknown representation is *SAX*. It stands for 'Symbolic Aggregate Approximation' and was invented by Eamonn Keogh and Jessica Lin in 2002 at the Riverside University of California [11] [12]. It converts the time series into a string of symbols according to intervals of values given by an algorithm called *PAA - Piecewise Aggregate Approximation*. It was chosen for this thesis because of the said advantages of symbolic representations and its simpleness, which also is a big advantage when working with different systems or differently equipped robots. The following sections are based on the work of Eamonn Keogh and Jessica Lin.

3.1 Normalisation

Comparing time series or parts of it only makes sense if they are in the same range. In Figure 3.1, for example, two time series are shown, which both describe the movement of a hand. Although they represent the same movement, in this case waving, their values seem to be very different according to scale. The reason for this is that the Kinect sensor provides absolute data for positions. So by waving in two different positions results in time series useless to compare.

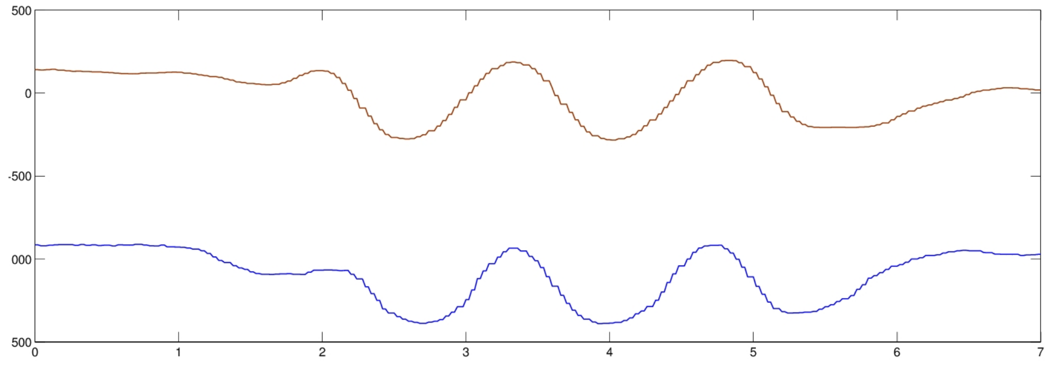

Figure 3.1: *Two time series, which show a similar hand waving, but were recorded in different positions.*

To make time series comparable it is necessary to normalise them. This is done in two steps:

- Subtract the mean of the time series from every value.
- Divide every value by the standard deviation of the resulting time series.

This produces a series with a mean of zero and a standard deviation of one. So both, offsets and amplitudes are aligned as demonstrated in Figure 3.2.

3.2 PAA - Piecewise Aggregate Approximation

To represent a time series via SAX it needs to be approximated by PAA after it was normalised.

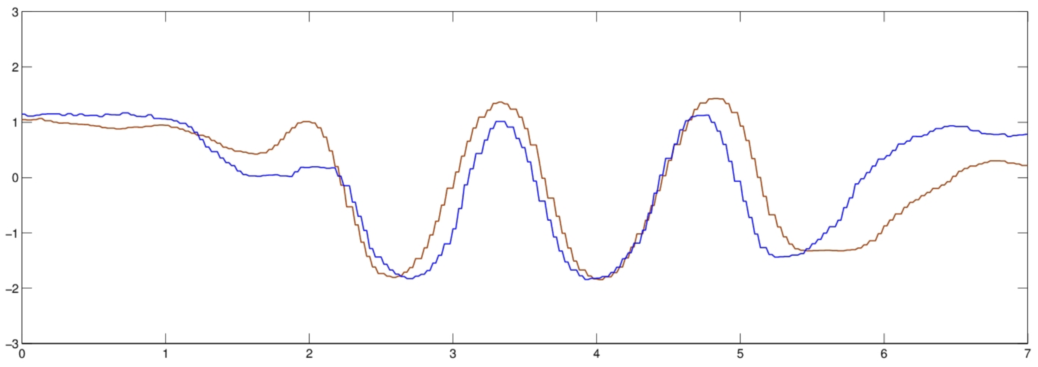

Figure 3.2: *The same time series as in Figure 3.1, but both of them are normalised hence comparable.*

Therefore a time series $T = t_1, \ldots, t_n$ of size n is transformed into a series $\overline{T}$ which consists of w *windows* $\overline{t_1}, ..., \overline{t_w}$. The value for each window $\overline{t_i}$ is given by

$$\overline{t_i} = \frac{w}{n} \sum_{j=\frac{n}{w}(i+1)-1}^{i\frac{w}{n}} t_j \tag{3.1}$$

This means you transform a time series of length n into a time series of length w, while the values of the new time series is the means of the corresponding windows. You can think of this transformation also as a representation where each segment is represented as a constant, the mean value, as you can see in Figure 3.3.

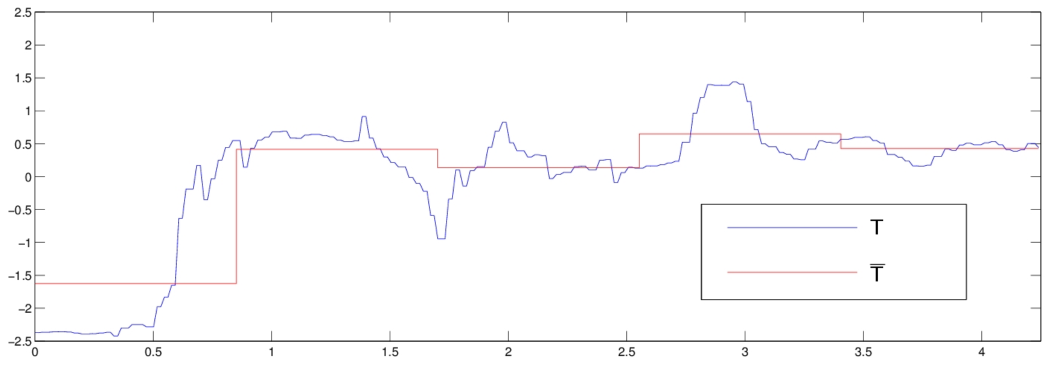

Figure 3.3: *A time series T of length 280 divided into 5 windows with PAA.*

The above shown algorithm creates windows of size $\frac{n}{w}$. This requires n to be divisible by w, otherwise there will an untreated rest. Of course, mostly this will not be the case. There are several ways to handle this problem.

One way is to just cut off all remaining points. While this is obviously the easiest way, it always results in a loss of information. If, for example, a time series with a length of 100 should be represented with 13 segments, 9 points of information are cut off and lost.

When avoiding any loss, there will be windows of different lengths. Apparently, the difference should never be greater than one point as not to falsify the information too much. The following algorithm fulfils the condition and is the one used in this thesis:

```
**for** $i = 1$ to $w$ **do**
    Length of $\overline{t_i} = \frac{n}{w}$ (cutting the decimal places)
    $n = n-$ Length of $\overline{t_i}$
    $w = w - 1$
**end for**
```

When dividing a time series of 100 points into 3 windows this way the window sizes are 33, 33, 34 and no information got lost.

3.3 SAX - Symbolic Aggregate Approximation

After applying the PAA transformation the time series are approximated by a step function consisting of w mean values. The next step is to discretise these mean values to a given set of symbols called *alphabet* with a *alphabet size a*. This thesis uses the first 10 letters of the classical Latin alphabet.

To reflect the time series characteristics at the best it is desirable to discretise them in such a way, that each symbol has the same probability to occur. After normalising a time series it has a Gaussian distribution, which makes it easy to set up the right intervals for discretisation. These intervals are defined by *breakpoints* $B = \beta_1, \ldots, \beta_{a-1}$. The area under a Gaussian curve between β_i and β_{i+1} must be $\frac{1}{a}$. The first interval is obviously from $-\infty$ to β_1 and the last one from β_{a-1} to $+\infty$. The exact values according to the alphabet size a can be looked up in a statistical table. Table 3.1 shows the values of all breakpoints for alphabet sizes from 3 to 10.

	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
β_3	-	0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
β_4	-	-	0.84	0.43	0.18	0	-0.14	-0.25
β_5	-	-	-	0.97	0.57	0.32	0.14	0
β_6	-	-	-	-	1.07	0.67	0.43	0.25
β_7	-	-	-	-	-	1.15	0.76	0.52
β_8	-	-	-	-	-	-	1.22	0.84
β_9	-	-	-	-	-	-	-	1.28

Table 3.1: The breakpoints which define the intervals for the discretisation of the different mean values for alphabet sizes from 3 to 10.

Now all the mean values given by the PAA transformation are converted to the symbol corresponding to the interval they are lying in. So all points of the original time series, whose windows mean value is below β_1 are now represented by the first symbol in the alphabet, here a. All points, which were represented by mean values from β_1 to β_2 are now represented by the second symbol, here b, and so on. The resulting list of symbols is called *word*, which is, obviously, of length w. Figures 3.4, 3.5 and 3.6 show the whole process from the original time series, over normalisation and PAA to the final representation by SAX.

Most motions consist of movements from different body parts. Consequently, recording motions leads to more than one time series. Furthermore every movement is made in the three dimensional space. So, for example, if a motion is recorded by observing the movements of both hands and both elbows you get a total amount of 12 time series. For easier demonstration and computing, the SAX representations of these time series, the *word* so to speak, are bundled in to a matrix.

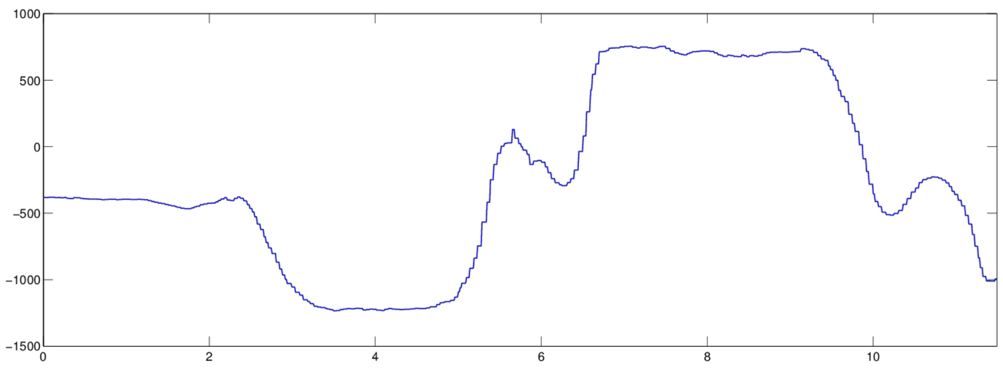

Figure 3.4: *The originally recorded time series.*

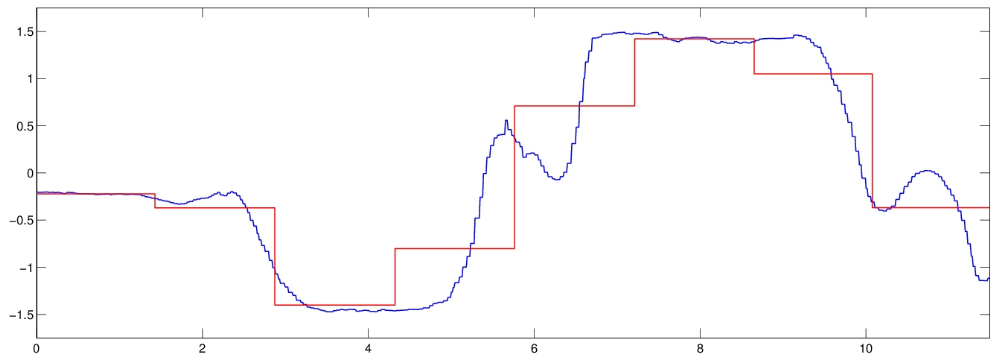

Figure 3.5: *The time series of Figure 3.4 normalised and represented by PAA with $w = 8$ windows.*

Here is an example of such a matrix. It represents the movements of the right hand and elbow while waving, with $w = 8$ and $a = 5$:

$$\begin{pmatrix} a & a & b & e & d & e & e & c \\ d & c & d & d & b & b & d & a \\ c & c & c & b & b & a & c & c \\ a & b & e & e & b & b & b & c \\ a & c & d & c & c & a & e & d \\ e & c & a & d & e & e & d & a \end{pmatrix}$$

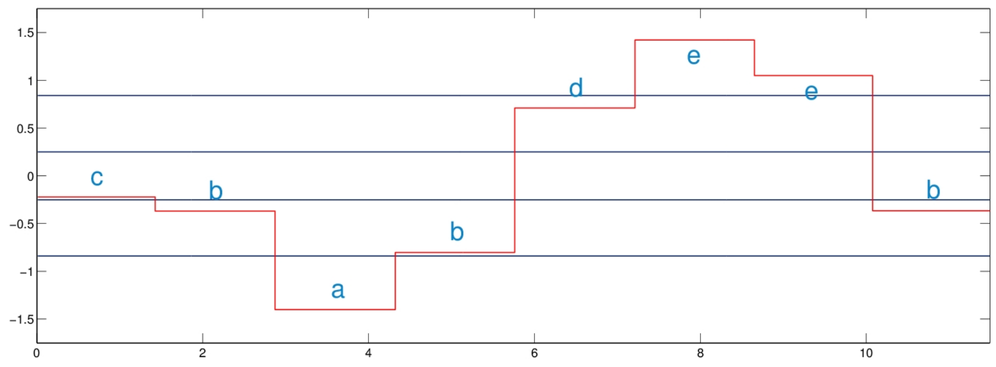

Figure 3.6: *This plot shows the breakpoints and the discretisation by SAX of the time series in Figure 3.5. As a result the time series is represented by the word 'cbabdeeb' using an alphabet size of $a = 5$.*

4 Measuring Distances

When attempting to recognise motions the currently observed movements and therefore their time series are compared to the saved motion primitives and their corresponding time series. Regardless of whether the time series are represented or transformed in any way or not, in order to compare them, it is necessary to measure a distance between them. The lower the distance between two time series, the more they are alike, and so are the motions they stand for. Thus, analysing the observed movements means to calculate the distance between them and the saved time series.

There are different ways to use the distance in order to check if a motion is recognised. One way is to take the saved motion with the smallest distance as the recognised one. The problem here is obviously that it will always recognise a motion, no matter if the observed motion is represented by the saved motion primitives or not. To avoid this problem it is necessary to define a limit. As soon as the distance falls below this limit the motion is counted as recognized. This is probably the most common way. It is also possible to set a limit between the different distances. A motion is then regarded as recognized when it is has the smallest distance of all motions and the difference to the second smallest distance is greater than the set limit.

The most common distance measure is the *Euclidean distance*, also known as Euclidean metric. Given two time series $T_1 = (t_{11}, \ldots, t_{1n})$ and $T_2 = (t_{21}, \ldots, t_{2n})$ with length n in its original status, the Euclidean distance between them is defined by the following equation (4.1) [11].

$$EuclDist(T_1, T_2) = \sqrt{\sum_{i=1}^{n} (t_{1i} - t_{2i})^2} \tag{4.1}$$

In the experiments of this thesis the time series are represented with the PAA and SAX algorithms as explained in the Sections 3.2 and 3.3. When representing a time series $T_1 = (t_{11}, \ldots, t_{1n})$ with PAA, resulting in $\overline{T_1} = (\overline{t_{11}}, \ldots, \overline{t_{1w}})$, the Euclidean distance of Equation (4.1) is easily adjusted.

$$EuclDist(\overline{T_1}, \overline{T_2}) = \sqrt{\frac{n}{w}} \cdot \sqrt{\sum_{i=1}^{w} (\overline{t_{1i}} - \overline{t_{2i}})^2} \tag{4.2}$$

This time it is only necessary to sum up the differences between the mean values of the windows, in which the time series are segmented. To balance the reduction, the size of the windows $\frac{n}{w}$ is taken account of.

The next step is to find a way to measure the distance after the time series are represented by the SAX algorithm. Actually, the calculation for the overall distance stays the same. The question is how the distances between the symbols are defined.

Jessica Lin and Eamonn Keogh, the developers of SAX, defined a distance measure called *MINDIST function* [11]. As before the major calculation is still the same as in Equation (4.2). The distances between each of the symbols are stored in look-up tables. For each alphabet size a a look-up table has to be made once and can then be stored and used easily. Figure 4.1 shows the look-up table for alphabet size 5. The distance between, e.g. '*a*' and '*d*' is stored in cell (1,4). The distance values depend on the breakpoints explained in Section 3.3. Equation (4.3) shows how the value for each cell is calculated by Lin and Keogh.

$$distance(i,j) = \begin{cases} 0, & |i-j| \leq 1 \\ \beta_{max(i,j)-1} - \beta_{min(i,j)}, & \text{else} \end{cases} \tag{4.3}$$

	1	**2**	**3**	**4**	**5**
1	0	0	0.59	1.09	1.68
2	0	0	0	0.5	1.09
3	0.59	0	0	0	0.59
4	1.09	0.5	0	0	0
5	1.68	1.09	0.59	0	0

Table 4.1: The look-up table used by the MINDIST function for alphabet size $a = 5$. The distance between two symbols can easily be read off at the corresponding row and column. For example, the distance between 'a' and 'd', using the Latin alphabet, is 1.09.

Altogether, Equation (4.4) shows the MINDIST function for two represented time series $\hat{T}_1$ and $\hat{T}_2$.

$$MINDIST(\hat{T}_1, \hat{T}_2) = \sqrt{\frac{n}{w}} \cdot \sqrt{\sum_{i=1}^{w} (distance(\hat{t}_{1i}, \hat{t}_{2i}))^2} \tag{4.4}$$

The advantage of the MINDIST function is the fact that it approximates and lower bounds the original Euclidean distance as in Equation (4.1), which is proven by Jessica Lin in [12]. This means, that the distance calculated by the MINDIST function is always smaller than the Euclidean distance of the original time series. While this is generally an important feature, because it allows applying many data mining algorithms to the represented time series producing the same results as the algorithms would for the original time series, it is no advantage in this work.

On the other hand, there are two problems working with the MINDIST function as in Equation (4.4). First of all, it assumes that the compared time series are both of the same length n. As described later in Chapter 6, the saved time series are cut to the essential, thus they are of different lengths. Furthermore the distance received from the MINDIST function depends on the alphabet size a and the length of the time series n. The highest possible distance increases with the length of the original time series and the size of the alphabet. In general, this does not have to be a problem, but the fact that the highest possible distance varies makes it impossible to set a fix recognition limit.

As a comparison another algorithm will also be used in the experiments in Chapter 6. It can be considered as the average euclidean distance between two motions. There is also a slight difference at the discretisation into the symbols. Instead of working with breakpoints according to the Gaussian distribution as used in the normal SAX algorithm, this time the symbols are assigned using equal steps. After the normalisation the range between the lowest and the highest value of the time series is split into a equal sized sections as it is done with the time axis by dividing it into the windows. According to the interval its mean value lies in, each window is assigned a whole number between 1 and a.

As said previously, a motion often consists of the movements from more than one part of the body and therefore of several time series. They are now seen as one time series in a $s+1$ dimensional space. Hence, every window and its s assigned values stands for a point in this space. When comparing two motions and their time series with the alternative way, the euclidean distance between each of those points is calculated. The mean value of these single distances is then taken as the distance between the two motions. The complete calculation is shown in (4.5).

$$MeanEuclDist(\tilde{T},\tilde{U}) = \frac{1}{(a-1)\cdot w\cdot\sqrt{s}}\cdot\sum_{j=1}^{w}\sqrt{\sum_{i=1}^{s}(\tilde{t}_{ij}-\tilde{u}_{ij})^2} \tag{4.5}$$

The alphabet size a and the number of time series s are taken account of, always resulting in distances between 0 and 1, where 0 means the transformed time series are identical.

5 Markov Models

As explained in the Introduction, a gesture can be seen as a sequences of several smaller movements. For example, a simple waving with one hand could consist of three movements: first the hand is raised above the head. Then the hand is moved to the left and to the right. At the end it is moved back down to the initial position.

To clarify which kind of movement is meant, two different words are selected and used from now on. The complete 'bigger' movement is referred to as *gesture*. In the example the whole hand waving thus is a gesture. The 'smaller' movements are referred to as *motions* or *motion primitives*. The hand raising and the left-right movements would be motions. So every gesture consists of several motions. A set of motions is recorded, which can then be combined to different gestures. Hence, it is possible that two gestures consist of the same motions, but in a different order.

Most procedures in nature, technology or economy are not deterministic or at least have some non-deterministic parts in it. One way to describe these systems are Markov Models, statistical models named after the Russian mathematician Andrey Markov [14]. There are mainly two Markov Models, Markov Chains and Hidden Markov Models, both are explained in the following sections. Hidden Markov Models are a extension of the Markov Chains and are a very popular instrument for pattern recognition, especially for speech recognition. The following sections are based on the work of Lawrence R. Rabiner [14].

5.1 Markov Chains

Markov Chains are discrete models to describe stochastic processes. It is assumed that the process proceeds in discrete time steps $t_i, i = 1, \dots, T$. In each of these time steps, the process is in a state $x_i \in S$, where $S = s_1, \dots, s_N$ is the finite set of all states resulting in a series of state transitions $X = (x_1, \dots, x_T)$. The transitions from one state to another are described using probabilities. One important rule for all Markov Models is that the probability that the system will be in a certain state at the next time step only depends on the present state:

$$P(x_{t+1} = s_i \mid x_t = s_j, x_{t-1} = s_k, \dots) = P(x_{t+1} = s_i \mid x_t = s_j) \tag{5.1}$$

This attribute of a stochastic process to not have a memory is also known as Markov Property.

Another rule of Markov Models is that they are time invariant. This means the probability to change from one state to another is the same for all time steps:

$$P(x_{t+1} = s_i \mid x_t = s_j) = P(x_t = s_i \mid x_{t-1} = s_j) \ \forall \ t \tag{5.2}$$

With these rules you get a $N \times N$ state transition matrix A with the following coefficients:

$$a_{ij} = P(x_{t+1} = s_j \mid x_t = s_i), \quad 1 \leq i, j \leq n \tag{5.3}$$

The following rules must apply for all of them:

$$0 \leq a_{ij} \leq 1 \tag{5.4}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{5.5}$$

So this matrix describes all the probabilities with which the system will change its state independent of time and previous states. As you can see in Equation (5.4), it can also be impossible to change from one specific state to another. In that case, a_{ij} is zero. The question remains which state the system is in at the beginning. For this purpose you declare a $N \times 1$ vector Π with the probabilities for every state to be the initial state:

$$\pi_i = P(x_0 = s_i) \tag{5.6}$$

$$0 \leq \pi_i \leq 1 \tag{5.7}$$

$$\sum_{i=1}^{N} \pi_i = 1 \tag{5.8}$$

Given such a system you can observe the sequence of state transitions. Let's say your Markov Chain describes the movements of a person and every state stands for a certain position. For example the state s_1 means the person is standing normally with both arms hanging down. s_2 means both arms are raised to the top and s_3 that the arms are stretched out to the side. This results in a set of possible states $S = \{s_1, s_2, s_3\}$. In this scenario A and Π could be as follows:

$$A = \begin{pmatrix} 0.6 & 0.3 & 0.1 \\ 0.5 & 0.3 & 0.2 \\ 0.6 & 0.1 & 0.3 \end{pmatrix}$$

$$\Pi = \begin{pmatrix} 0.8 \\ 0.1 \\ 0.1 \end{pmatrix}$$

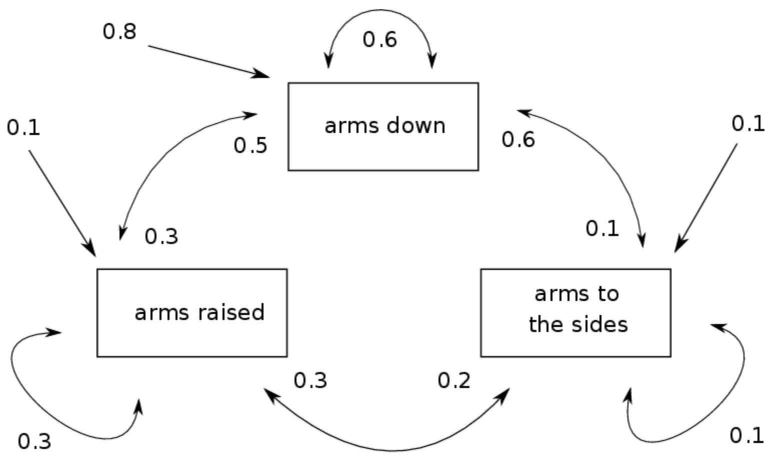

Figure 5.1: *The whole chain with its three states, the transition and initial probabilities.*

Hence, in this example, it is more likely that the person starts and moves to the normal position with no arm raised. The whole chain is shown in Figure 5.1.

In such scenarios one is often interested in the probability that the positions follow in a certain order like arms down, arms raised, arms raised, arms down, arms raised, arms to the side. Formally we want to compute the probability for the sequence $X = (s_1, s_2, s_2, s_1, s_2, s_3)$.

$$\begin{aligned}
P(X) &= P(s_1, s_2, s_2, s_1, s_2, s_3 \mid A, \Pi) \\
&= P(s_1) \cdot P(s_2 \mid s_1) \cdot P(s_2 \mid s_2) \cdot P(s_1 \mid s_2) \cdot P(s_2 \mid s_1) \cdot P(s_3 \mid s_2) \\
&= \pi_1 \cdot a_{12} \cdot a_{22} \cdot a_{21} \cdot a_{12} \cdot a_{23} \\
&= 0.8 \cdot 0.3 \cdot 0.3 \cdot 0.5 \cdot 0.3 \cdot 0.2 \\
&= 0.00216
\end{aligned}$$

5.2 Hidden Markov Models

The Markov Chain from the previous section assumes that you can always observe the current state. In Hidden Markov Models (HMM), as the name implies, the states are hidden. This means that you never know in which state the system is or was. Instead of following directly the states and the transitions between them, the system produces an observable output o_t at every time step t. The series of outputs over a time T is called the observation $O = (o_1, \ldots, o_T)$. These outputs are again linked to a stochastic process. Each state has a probability to produce a certain output. One task is to estimate the current state with the help of a given observation and the probabilities to produce it, which will be explained later. You can think of the Hidden Markov Model as a stochastic model with two linked layers, in which one is hidden and therefore not observable.

Each HMM λ is defined by a 5-tuple (S, A, Π, K, B). Like before, S is the set of states, A is the transition matrix giving the probabilities how to change the states and Π is the set of probabilities for each state to be the initial one. K is the set of all possible outputs $\{k_1, \ldots, k_M\}$ the system can produce. B is a $N \times M$ matrix, which contains the probabilities for each state s_i to produce a certain output $k_j \in K$.

$$\begin{aligned} b_{ij} &= P(o_t = k_j \mid x_t = s_i), \quad \forall t, \quad 1 \leq i \leq N, \quad 1 \leq j \leq M \\ &= b_i(k_j) \end{aligned} \tag{5.9}$$

The second notation is sometimes used for the sake of clarity. The coefficients of B are subject to the following conditions:

$$0 \leq b_{ij} \leq 1 \tag{5.10}$$

$$\sum_{j=1}^{M} b_{ij} = 1 \tag{5.11}$$

Back to the example from Section 5.1, again the movements of a person are recorded. But this time it is impossible to know the whole position of the body or the gesture the person is doing. Instead, the position of single bodyparts like hands or elbows are the only information. The, now hidden, states S are still '*standing normally with arms down*', '*arms raised*' and '*arms stretched to the sides*'. The possible outputs K are k_1: '*hands are down*', k_2: '*hands are raised*', k_3: '*elbows are down*' and k_4: '*elbows are raised*'. In Figure 5.2 you can see an illustration of the whole HMM with its three hidden states, four possible outputs and all probabilities. For this example, the matrix B looks like this:

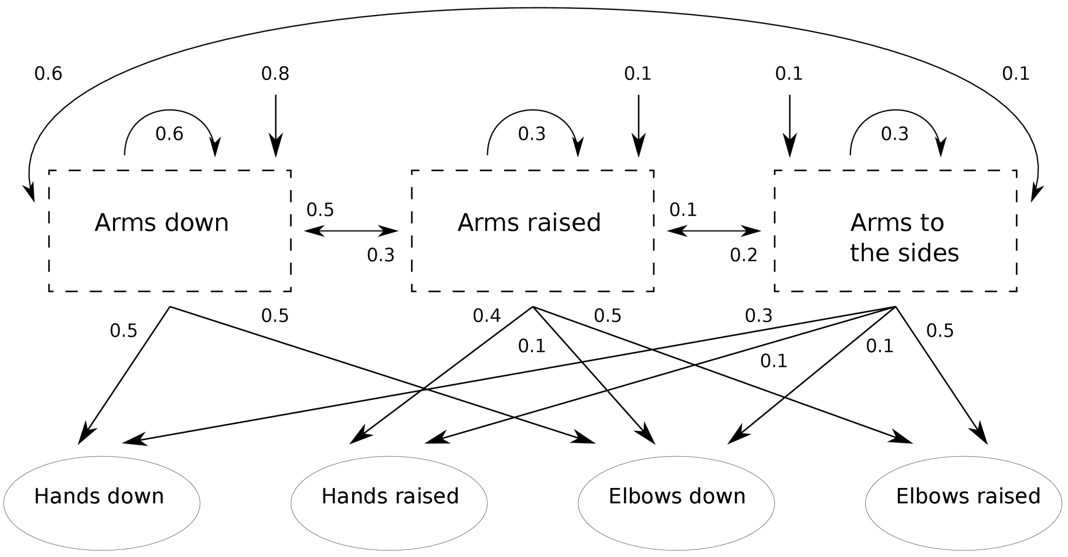

Figure 5.2: *The complete HMM with its three hidden states, four possible outputs and probabilities for transition, output and initial state.*

$$B = \begin{pmatrix} 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.4 & 0.1 & 0.5 \\ 0.3 & 0.1 & 0.1 & 0.5 \end{pmatrix}$$

So the probability that the system will produce the output k_2 '*hands are raised*' while at state s_2 '*arms are raised*' is 0.4. Also, it is impossible that the elbows are raised when in state '*arms hanging down*'.

To use HMMs for gesture recognition in praxis, each known gesture is represented by a certain HMM λ_i. Recording the movements of a person results in an observation O as described above. The object is then to calculate which HMM has the highest possibility to produce these series of outputs. Another task is to train the HMMs to maximize their probabilities for relevant observations. Solutions to these problems are given in the next sections.

5.3 The Forward-Backward Algorithm

Because the states in a HMM are unknown, one important task is to get the probability to be in a certain state at a certain time step, given only the observation. Another important information is the probability an observation O will be made given the current state. For the first problem the solution is the Forward Procedure. For the second one you can use the Backward Procedure. Together they are used in the Baum-Welch-Algorithm to train HMMs.

5.3.1 The Forward Procedure

The Forward Procedure calculates the probability with which the system will produce an observation O and is in a certain state $x_t = s_i$ afterwards. This probability is called the *forward variable* $\alpha_t(O, i)$:

$$\alpha_t(O, i) = P(O = (o_1, \ldots, o_t) \mid \lambda, x_t = s_i) \tag{5.12}$$

The forward variable can be calculated recursively.

- Initialization:

$$\alpha_1(O, i) = \pi_i \cdot b_i(o_1) \tag{5.13}$$

- Induction:

$$\alpha_t(O, i) = \sum_{j=1}^{N} \alpha_{t-1}(O, j) \cdot a_{ji} \cdot b_i(o_{t-1}) \tag{5.14}$$

The forwards variable is used to get the overall probability that a HMM produces a certain observation, which is one of the main tasks when working with HMMs for pattern recognition. It is the sum over the forward variables of the current time step for all states:

$$P(O = (o_1, \dots, o_t) \mid \lambda) = \sum_{i=1}^{N} \alpha_t(O, i) \tag{5.15}$$

5.3.2 The Backward Procedure

The Backward Procedure is the counterpart to the Forward Algorithm. Instead of calculating the probability that the HMM has produced a certain observation O when ending in state s_i, it leads to the probability the HMM will produce O after it has been in s_i. This time it is called backward variable $\beta_t(O, i)$.

$$\beta_t(O, i) = P(O = (o_{t+1}, \dots, o_T) \mid \lambda, x_t = s_i) \tag{5.16}$$

Again it's calculated by recursion.

- Initialization:

$$\beta_T(O, i) = 1 \tag{5.17}$$

- Induction:

$$\beta_t(O, i) = \sum_{j=1}^{N} \beta_{t+1}(O, j) \cdot a_{ij} \cdot b_j(o_{t+1}) \tag{5.18}$$

5.4 The Baum-Welch-Algorithm

One important object when dealing with HMMs is to train them. This means to adjust the HMM to a given observation so it is more likely to produce it. Say, you have five movements you want to be represented by HMMs. First you create five (identical) HMMs. You then train each HMM with a set of versions of the corresponding movement and their resulting observations. For example, using the same movement done by different persons. After the training each HMM should be more likely to produce these observations, formally $P(O, \lambda^i) \leq P(O, \overline{\lambda^{i+1}})$, and is ready to be used for gesture recognition. For this task, the Baum-Welch algorithm is used and explained below [13][14].

The forward and backward variables are used to calculate more useful variables. One is the probability that the system is in a specific state at a certain point of time and is called $\gamma_t(O, i)$.

$$\begin{aligned}\gamma_t(O, i) &= P(x_t = s_i \mid O, \lambda) \\ &= \frac{\alpha_t(O, i) \cdot \beta_t(O, i)}{P(O \mid \lambda)} = \frac{\alpha_t(O, i) \cdot \beta_t(O, i)}{\sum_{j=1}^{N} \alpha_t(O, j) \cdot \beta_t(O, j)}\end{aligned} \tag{5.19}$$

One step further, you need to know how likely it is that the transition between two specific states happens at a certain time step. In other words, you want to know the probability, called $\delta_t(O, i, j)$ that $x_t = s_i$ and $x_{t+1} = s_j$ for a given observation $O = (o_1, \ldots, o_T)$.

$$\begin{aligned}\delta_t(O, i, j) &= P(x_t = s_i, x_{t+1} = s_j \mid O = (o_1, \ldots, o_T), \lambda) \\ &= \frac{\alpha_t(O, i) \cdot a_{ij} \cdot b_i(o_{t+1}) \cdot \beta_{t+1}(O, j)}{\sum_{u=1}^{N} \sum_{v=1}^{N} \alpha_t(O, u) \cdot a_{uv} \cdot b_v(o_{t+1}) \cdot \beta_{t+1}(O, v)}\end{aligned} \tag{5.20}$$

With these formulas, it is possible to calculate the estimated numbers of transitions between the states for a given observation O. $\gamma_t\{O, i\}$ has been declared as the probability to be in state s_i at the time step t when the model produces the observation O. Summing up the probability to be in a state over the whole time T gives the estimation how often the model will be in that state. Leaving out $T-1$ therefore gives the estimated number of transitions from that state:

$$\sum_{t=1}^{T-1} \gamma_t(O,i) \tag{5.21}$$

When doing the same with $\delta_t(O,i,j)$ instead of $\gamma_t(O,i,j)$ you get an even more specific estimation. Not only do you get the estimated number of any transition from that state s_i, but the number of transitions to a certain state s_j:

$$\sum_{t=1}^{T-1} \delta_t(O,i,j) \tag{5.22}$$

The task to train a HMM means nothing else than to redefine or optimize the values for A,Π and B. The idea is to re-estimate them with their probabilities for a given training observation. For π_i, representing the starting probability, this means, its new value is the probability that state s_i will be at the beginning of the state series when producing O.

$$\overline{\pi_i} = \gamma_1(O,i) \tag{5.23}$$

A holds the probabilities for the state transitions. Hence, the re-estimation for each coefficient a_{ij} is the expected number of that transition regarding O divided by the expected number of any transition from s_i. This makes sure that $0 \leq a_{ij} \leq 1$ is still true.

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \delta_t(O,i,j)}{\sum_{t=1}^{T-1} \gamma_t(O,i)} \tag{5.24}$$

It is nearly the same for B. The new value for b_{ij} is the expected number of times the model will be in state s_i producing the output k_j divided by the expected number of times the model actually will be in that state.

$$\overline{b_{ij}} = \frac{\sum_{t=1}^{T} \gamma_t(O,i) \cdot \kappa(o_t,k_j)}{\sum_{t=1}^{T} \gamma_t(O,i)} \tag{5.25a}$$

$$\kappa(o_t,k_j) = \begin{cases} 1, & o_t = k_j \\ 0, & \text{else} \end{cases} \tag{5.25b}$$

Continuing on the example from the sections before, the following calculations show the Baum-Welch-Algorithm in praxis and its effects on the HMM. The sets of states S and possible outputs K are the same. For A, Π and B random values were chosen to create the first version λ^1 of the HMM:

$$A^1 = \begin{pmatrix} 0.42 & 0.17 & 0.41 \\ 0.04 & 0.71 & 0.25 \\ 0.44 & 0.03 & 0.53 \end{pmatrix}$$

$$\Pi^1 = \begin{pmatrix} 0.05 \\ 0.77 \\ 0.18 \end{pmatrix}$$

$$B^1 = \begin{pmatrix} 0.04 & 0.45 & 0.41 & 0.1 \\ 0.09 & 0.14 & 0.65 & 0.12 \\ 0.34 & 0.22 & 0.11 & 0.33 \end{pmatrix}$$

The represented motion shall be: *'hands down', 'elbows raised', 'hands raised', 'elbows raised', 'hands down'.* So the training observation O is $(k_1, k_4, k_2, k_4, k_1)$ and $T = 5$. The probability to produce this observation with the untrained HMM is

$$P(O \mid \lambda^1) = \sum_{i=1}^{3} \alpha_5(O, i)$$

$$= 2.35 \cdot 10^{-5} + 1.71 \cdot 10^{-5} + 24.20 \cdot 10^{-5}$$

$$= 2.83 \cdot 10^{-4}$$

After one training cycle the new values for A, Π and B are:

$$A^2 = \begin{pmatrix} 0.235447 & 0.0798623 & 0.68469 \\ 0.0420446 & 0.379709 & 0.578246 \\ 0.276272 & 0.0109724 & 0.712756 \end{pmatrix}$$

$$\Pi^2 = \begin{pmatrix} 0.0163737 \\ 0.391732 \\ 0.591895 \end{pmatrix}$$

$$B^2 = \begin{pmatrix} 0.107455 & 0.570599 & 0 & 0.321946 \\ 0.605507 & 0.107608 & 0 & 0.286885 \\ 0.435321 & 0.117557 & 0 & 0.447122 \end{pmatrix}$$

The probabilities to output k_3 is now zero for all states, which is obvious, because it does not appear in the training observation. The new probability to produce the training observation is now:

$$P(O \mid \lambda^2) = \sum_{i=1}^{3} \alpha_5(O, i)$$

$$= 5.00 \cdot 10^{-4} + 4.50 \cdot 10^{-4} + 5.50 \cdot 10^{-3}$$

$$= 6.45 \cdot 10^{-3}$$

Apparently, the HMM represents the observation much better. Below the probabilities after more training cycles.

$$\begin{aligned}
P(O \mid \lambda^3) &= 7.73 \cdot 10^{-3} \\
P(O \mid \lambda^4) &= 0.0106 \\
P(O \mid \lambda^5) &= 0.0165 \\
P(O \mid \lambda^6) &= 0.0270 \\
P(O \mid \lambda^7) &= 0.0481 \\
P(O \mid \lambda^8) &= 0.0953 \\
P(O \mid \lambda^9) &= 0.182 \\
P(O \mid \lambda^{10}) &= 0.243 \\
P(O \mid \lambda^{100}) &= 0.25
\end{aligned}$$

5.5 Zero Entries and Training Sets

An important property of the Baum-Welch algorithm is that every transition probability or output probability that once is set to zero remains zero. While this can be an useful feature as described in the next section, it can also produce problems with the training of the model. In the previously used example it has three states and four possible outputs. Initially every entry in the matrices A and B has been greater than zero. When the HMM has been trained in section 5.4, one possible output, *'Elbows down'*, didn't appear in the training observation. This led to the fact that the probability to produce this output was set to zero for every state and it was impossible for the output to occur after the training. This effect can cause problems when working with the HMMs as it is done in this thesis. Once created and saved it should be possible to train the model whenever wanted. For example, a person is doing the gesture the HMM stands for and thus creates an observation sequence using it as a training observation to train the HMM. Afterwards the person does the gesture again, but this time produces a different observation, which is then used for the training. This procedure is then done several times or, for example, by different people to increase the probability to detect a gesture when it's done with minor changes. Assuming the first training observation doesn't contain a certain output and its probability to occur is therefore set to zero, as described above, it is then impossible to 'reactivate' this ouput, no matter if the following training observations contain it or not. This can obviously result in a poorly trained HMM, thus in a bad gesture recognition.

To avoid this problem all L training observations are bundled to a set $V = O_1, \ldots, O_L$. Instead of training the HMM with different observations one after another, it is trained with the complete set V at once. Whenever the model should be trained, it is first reset its initial status, as it was before it has been trained the first time. After adding the new observation to the training set, the HMM is then trained. Training with the set is very similiar to the single training process. Every training observation $O_l \in V$ has a weighting c_l. In this thesis all weightings are first 1. Instead of adding observations to training set, when they are already included, their weighting is increased by 1. This leads to the same results, but has lower computational costs. Below are the formulas of the Baum-Welch algorithm adapted for the work with a training set.

$$\overline{\pi_i} = \frac{\sum_{l=1}^{L} \gamma_1(O_l, i) \cdot c_l}{\sum_{l=1}^{L} c_l} \tag{5.26}$$

$$\overline{a_{ij}} = \frac{\sum_{l=1}^{L} (\sum_{t=1}^{T-1} \delta_t(O_l, i, j)) \cdot c_l}{\sum_{l=1}^{L} (\sum_{t=1}^{T-1} \gamma_t(O_l, i)) \cdot c_l} \tag{5.27}$$

$$\overline{b_{ij}} = \frac{\sum_{l=1}^{L} (\sum_{t=1}^{T} \gamma_t(O_l, i) \cdot \kappa(o_{l,t}, k_j)) \cdot c_l}{\sum_{l=1}^{L} (\sum_{t=1}^{T} \gamma_t(O_l, i)) \cdot c_l} \tag{5.28a}$$

$$\kappa(o_{l,t}, k_j) = \begin{cases} 1, & o_{l,t} = k_j \\ 0, & \text{else} \end{cases} \tag{5.28b}$$

The new values for A, B and π still fulfil the conditions. Another advantage of using a training set is that it doesn't matter in which order the observations have been made.

5.6 Left-Right Models

In the Hidden Markov Model described until now, it has been assumed that transitions can be made from every state to any other state. HMMs in which every state can be reached by any other state in a finite number of transitions are called fully connected or ergodic. Conditions (5.4) and (5.7) show that transition probabilities can also be zero, so the transition doesn't exist or can't be taken. Hence, it is possible to set certain entries in A or π initially to zero. As described in the previous section, zero entries will remain zero when working with the Baum-Welch algorithm. This allows to create special variants of HMMs. The most used one, besides the fully connected, is the Left-Right Model. In this HMM transitions can only be made from states with lower indices to states with higher indices. Ensured by condition (5.29), it results in upper-triangular transition probability matrices such as the following one with $N = 5$:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ 0 & a_{22} & a_{23} & a_{24} & a_{25} \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{pmatrix}$$

$$a_{ij} = 0, \quad i \geq j \tag{5.29}$$

Since the purpose of the Left-Right Model is a state sequence through all states, it is necessary to start in s_1.

$$\pi(i) = \begin{cases} 1, & i = 1 \\ 0, & i > 1 \end{cases} \tag{5.30}$$

The conditions can even be stricter. For example, $a_{ij} = 0, i = j, i \neq N$, so it is impossible for the HMM to stay in the current state if it is not the last one. Another popular one is to set a maximum step range Δ, shown in (5.31). The following A matrix shows this with $N = 5$ and $\Delta = 2$.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 \\ 0 & a_{22} & a_{23} & a_{24} & 0 \\ 0 & 0 & a_{33} & a_{34} & a_{35} \\ 0 & 0 & 0 & a_{44} & a_{45} \\ 0 & 0 & 0 & 0 & a_{55} \end{pmatrix}$$

$$a_{ij} = 0, \quad i + \Delta \leq j \tag{5.31}$$

Right-Left Models often reach the training target faster than fully connected HMMs, thus they are in many cases the better solution. Those cases can often be recognised, when a fully connected HMM tends to the Right-Left form after some training iterations. Starting with a Right-Left Model right from the beginning can then save some of those iterations.

6 Experimental Results

6.1 Experiment procedure

In this chapter the results of the experiments for time series representation and gesture recognition will be shown and analysed. As said before, the whole process is split into two parts.

The procedure for the experiments was always the same. First the motion primitives were recorded and saved. Second the active movements were compared to the saved motions to check the persons movements for known primitives. All recognized motions were stored in a list. After performing a complete gesture, consisting of several saved motion primitives, the resulting list of detected motions were used as an observation sequence to create a Hidden Markov Model representing the gesture.

Each motion primitive was recorded and saved separately. At all recordings and experiments the three-dimensional positions of the following body parts were used:

- Head
- Neck
- Right Shoulder
- Right Elbow
- Right Hand
- Left Shoulder
- Left Elbow

- Left Hand
- Torso (Centre of the upper body)

With three time series per joint the algorithms worked with a total of 27 time series. These time series were cut leaving only the essential part, which represents the motion. Although it is normally the aim to save the primitives in their abstract form, their time series were saved in their raw status. This simplified the comparison between the actual movements and the saved motions with, for example, different number of windows or different alphabet sizes. Keeping the time series saved in their original status made it possible to reuse them with different settings. Before an experiment started, the saved time series were, of course, discretised, hence saving them in their raw status did not effect the results.

With all required motions saved, the actual recognition process could begin. While the person moved in front of the Kinect, its movements were compared to the saved motions. Therefore only the last parts of the time series of the active movements were used. These parts were of the same length, regarding to time, as the currently reviewed saved motion. The frequency with which the comparison was taking place depends on the length of one window. An example shall clarify this. The movements of a person shall be checked for a motion primitive. This motion is 1 second long. The number of windows, in this example, is 5, so one window of the motion is 0.2 seconds long. Now with a frequency of 0.2 seconds the last 1 second of the active time series are represented via SAX and then compared to the transformed time series of the saved motion. Hence, the movements of a person are compared to a quick motion more often than to a slow, long taking motion.

The comparison was done by calculating the distance as described in chapter 4. A motion counted as recognised when its distance to the current movements fell below a set limit and no other motion did as well. Thus, it was possible that a motion was recognised more than one time for the same movement, which can effect the overall process.

Every saved motion primitive was performed 50 times for every setting, noting the number of correct recognitions resulting in a recognition rate. Also, every motion was tested with both ways of discretisation. The SAX algorithm with the MINDIST function on the one hand and the alternative more simple way of the average euclidean distance on the other hand. The variant of the average euclidean distance was tested with each combination of 5, 10 and 15 windows and an alphabet size of 5, 10 and 15. The same applied to the SAX algorithm but without the alphabet size of 15, because it is not declared for alphabet sizes greater than 10 [12].

Afterwards 15 Hidden Markov Models have been created for each complex gesture. There were three groups of five HMMs, in the first group all models got trained with only one training observation, in the second the training set consisted of five observations and in the last group of ten. In each group the models are trained either 1,2,5,10 or 20 times. Again every gesture was performed 50 times. Whenever the probability of an HMM to produce the current observation was above a set limit and no HMM of another gesture did as well, the gesture counted as recognised.

6.2 The motion primitives and complex gestures

To get the best possibilities to analyse the algorithms the choice of the primitives should cover rather easy and rather hard ones to recognise. Also there should be short and longer ones and some should be similar to see if the algorithms can differ between them. All motions are about the handling of a bottle. Trying to cover all the said aspects in a small set of motions resulted in the primitives in Table 6.1.

The complex gestures are obviously sequences of the described motion primitives:

- Grab a bottle and a glass, fill the glass and put the bottle back:
  Grab - Pour - Put Down
- Pick up a bottle, open it and put it away:
  Pick Up - Open - Put Down
- Pick up a bottle, open it, get a glass, fill it and put the bottle away:
  Pick Up - Open - Pour - Put Down

Name	**Description**	**Length (time)**	**Length (points)**
Pick Up	The person picks up a bottle off the floor with its right hand.	2.12 s	140
Open	The person holds a bottle in its right hand and opens it.	3.12 s	205
Put Down	The person holds the bottle in its right hand. It is then put down on a table to the right.	2.93 s	194
Grab	The person reaches to its right to grab a bottle standing on a table.	1.95 s	130
Pour	The person holds a bottle in its right hand. It gets a glass from the left and pours the content of the bottle into it.	4.60 s	304

Table 6.1: The five different motion primitives, with which the algorithms have been tested.

6.3 Results

The results for the first part of the process are shown in Table 6.2.

First of all, the results for both algorithms are surprisingly good, considering that they are both quite simple. The Kinect and its software probably made a significant contribution, creating noiseless time series.

The only values that stand out are those for the 'Open' motion for both algorithms. There are two reasons for this: firstly, the movements of opening a bottle are very small and therefore the resulting time series were not very characteristic. Secondly, while opening a bottle the arms are in front of the body, sometimes even on top of each other from the point of view of the Kinect. This can lead to misinterpretations by the Kinect software and therewith to wrong time series with irregularly occurring extrema. Greater alphabets and higher number of windows appear to decrease these problems but can't eliminate them.

Although, the two motions 'Put Down' and 'Grab' are very similar, both algorithms

a	w	Pick Up	Open	Put Down	Grab	Pour
5	5	50 / 80	32 / 44	60 / 90	80 / 94	32 / 96
5	10	96 / 86	60 / 42	88 / 86	92 / 94	68 / 96
5	15	94 / 80	66 / 56	96 / 86	86 / 92	84 / 90
10	5	74 / 76	56 / 34	70 / 88	90 / 98	32 / 94
10	10	98 / 86	80 / 50	96 / 94	100 / 100	96 / 98
10	15	96 / 90	86 / 66	96 / 90	100 / 98	94 / 96
15	5	- / 74	- / 46	- / 90	- / 98	- / 96
15	10	- / 90	- / 50	- / 96	- / 96	- / 100
15	15	- / 86	- / 60	- / 94	- / 96	- / 96

Table 6.2: The recognition rates of the five motion primitives in percent with different number of windows and different alphabet sizes. The blue entries are the results of the MINDIST function and the brown entries are the results of the average euclidean distance variant.

could distinguish between them well. Especially the 'Grab' movement was recognised constantly very well. Also noticeable are the good rates of the 'Pour' motion, even though it is the longest and most complex motion of all. While this is true for the average euclidean distance for all settings, the MINDIST function only produces good recognition rates with a great alphabet and a higher number of windows. This fact also applies to the other motions. With 5 symbols and 5 windows, the MINDIST function is inferior to the average euclidean distance for all motions. The average euclidean distance is less dependent on the alphabet size and the number of windows, but doesn't produce as good and consistent results as the MINDIST function. Even greater alphabet sizes than 10 doesn't seem to make a difference.

Not shown in Table 6.2 are the effects of the window sizes. The higher the number of windows, the more often the recognition process takes place. For 15 windows the motion primitives nearly always were recognised more than one once at the same time. Obviously, this happens more often the shorter the motion primitives are. This also affects the overall procedure including the Hidden Markov Models. To get high recognition rates, but as few multiple recognitions as possible, for the second part of the tests the

MINDIST function with an alphabet size of 10 and 10 windows was chosen. The results for these experiments are shown in Table 6.3.

Training Set Size	**Training Iterations**	**Grab - Pour - Put Down**	**Pick Up - Open - Put Down**	**Pick Up - Open - Pour - Put Down**
1	1	30	4	18
1	2	62	8	22
1	5	24	0	36
1	10	18	0	4
1	20	8	0	0
5	1	34	12	20
5	2	58	8	32
5	5	88	8	80
5	10	94	52	42
5	20	94	38	34
10	1	52	10	28
10	2	66	12	48
10	5	100	10	90
10	10	100	68	96
10	20	100	40	96

Table 6.3: The results for the complete procedure with different number of training observations and iterations.

The results for the complete procedure are very diverse. Apparently, the number of training observations is very important. A low training set size and a high number of iterations produce poor results, because in that case, the HMM is not sensitive for observations that differ from the few training observations. Also noticeable are the constantly poor results for the second gesture. All three contained motion primitives are also part of the third gesture, which led to wrong recognitions between these two gestures. The poor recognition rate of the motion 'Open' is relevant at this as well.

6.4 Conclusion

The results showed, that time series representations are a good possibility to transform motion sequence information into platform independent information. It was possible to represent the multi-dimensional sensor signal with very little information and still maintain its particular characteristic. The simple SAX algorithm provided good results in recognition while keeping the computation costs and the needed storage low. It is, however, necessary to adjust its 'settings', the alphabet size and the number of windows, to the attributes, for example the length, of the used motion primitives. The differences between the MINDIST function and the average euclidean distance are small, whereby the former is more recommended, because it achieves slightly better and more secure results. The Hidden Markov Model is a good solution to recognise the sequences of the motion primitives, although it is very important to provide it with enough training observations. The Kinect sensor and its software creates reliable and noiseless signals as long as body parts do not overlap.

List of Figures

List of Tables

Bibliography

[1] http://www.roboearth.org/ , Last checked on March 2012.

[2] Waibel M.; Beetz M.; Civera J.; D'Andrea R.; Elfring J.; Galvez-Lopez D.; Haussermann K.; Janssen R.; Montiel J.M.M.; Perzylo A.; Schiessle B.; Tenorth M.; Zweigle O.; van de Molengraft R.: RoboEarth - A World Wide Web for Robots. IN: Robotics and Automation Magazine, IEEE, 2011.

[3] Wujanz, D.; Weisbrich, S.; Neitzel, F.: 3D-Mapping mit dem Microsoft® Kinect Sensor - erste Untersuchungsergebnisse. IN: Proceedings 10.Oldenburger 3D-Tage, 2011.

[4] http://www.microsoft.com/presspass/press/2010/mar10/03-31primesensepr.mspx , Last checked on March 2012.

[5] http://community.guinnessworldrecords.com/_Kinect-Confirmed-As-Fastest-Selling-Consumer-Electronics-Device/blog/3376939/7691.html , Last checked on March 2012.

[6] http://blogs.msdn.com/b/kinectforwindows/archive/2012/01/31/kinect-for-windows-is-now-available.aspx , Last checked on March 2012.

[7] Stowers, J.; Hayes, M.; Bainbridge-Smith, A.: Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor. IN: Mechatronics (ICM), IEEE International Conference, 2011.

[8] Lu Xia, Chia-Chih Chen, J. K. Aggarwal: Human Detection Using Depth Information by Kinect. 2011.

[9] http://openni.org/ , Last checked on March 2012.

[10] Matthias Greuter, Michael Rosenfelder, Michael Blaich, Oliver Bittel: Object Detection with the 3D-Sensor Kinect. IN: International Conference on Research and Education in Robotics, 2011.

[11] Jessica Lin, Eamonn Keogh, Stefano Lonardi and Bill Chiu: Symbolic Representation of Time Series, with Implications for Streaming Algorithms. IN: 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2003.

[12] Jessica Lin, Eamonn Keogh, Stefano Lonardi, Li Wei: Experiencing SAX: a Novel Symbolic Representation of Time Series. 2007.

[13] Leonard E. Baum, Ted Petrie, George Soules, Norman Weiss: A Maximization Technique occuring in the Statistical Analysis of Probabilistic Functions of Markov Chains. IN: The Annals of Mathematical Statistics, Vol. 41, No. 1, 1970.

[14] Lawrence R. Rabiner: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. IN: Proceedings of the IEEE, Vol. 77, No. 2, 1989.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Felix Brucker)