FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Kinect-enabled activity recognition of multiple human actors for a service robot

Sören Jentzsch

FAKULTÄT FÜR INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Kinect-enabled activity recognition of multiple human actors for a service robot

Kinect-basierte Aktivitätserkennung mehrerer menschlicher Akteure für einen Serviceroboter

Author:	Sören Jentzsch
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Dipl.-Ing. Claus Lenz
	Manuel Giuliani, M.Sc.
	Andre Gaschler, M.Sc.
Submission Date:	October 17, 2011

Ich versichere, dass ich diese Bachelorarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this bachelor's thesis only supported by declared resources.

München, den 17. Oktober 2011 __________________________________

Sören Jentzsch

Abstract

The purpose of this bachelor's thesis is to present a machine learning based approach to recognize several activities of persons interacting with a robot, based on the position of their joints. In order for service robots to understand human behavior and to interact with humans, this thesis investigates human-robot interaction within the JAMES project (Joint Action for Multimodal Embodied Social Systems) on the basis of a bar scenario, in which a robot bartender serves human actors. Motivating hidden Markov models (HMMs) to fulfill our activity recognition task, we first focus on the theory behind them. Whereas the joint data is obtained by the Kinect, a motion sensing device, this thesis further examines how to develop a software program to provide us with our final number of 2307 labeled activities we utilize to train and test our HMM. The results of the 695 labeled activities used for testing reveal that about 90% of these were correctly recognized. Considering activities which were recognized without occurrence in the test data, we obtain an accuracy of over 70%. As this performance is in the same order of magnitude as the training results, our approach for solving the activity recognition task in our bar scenario is fairly reliable and robust. The contribution of this thesis is to provide initial research in activity recognition within the JAMES project and hopefully to lay a foundation for further investigation and improvement in this area of research.

Contents

List of Figures

1 Introduction

Human-robot interaction is said to have a huge impact on our life as science advances. For now, robots, in this case rather called machines, mainly operate in environments excluding real interaction with humans. In order to achieve the change towards social robots, besides the shape of them, they have to reliably detect, track and understand the activities of humans and then react appropriately. As current research on human-robot interaction takes great steps forward, this thesis will try to contribute to the field of human activity recognition.

1.1 Motivation

The work of this thesis is based upon the goals of the JAMES project (Joint Action for Multimodal Embodied Social Systems) [28]. The JAMES project researches the area of social robots in terms of interacting with humans in a socially acceptable way, as stated by the robotics partner of the project, fortiss GmbH [8]. There are many tasks for a robot to become a social robot. For example, he has to understand what humans are saying, doing and, if possible, intend to do. Among tasks like motion planning, dialogue management and body language detection, the robot should be able to recognize the current activitiy of humans in his field of vision. The latter task lays the foundation for this thesis.

With this work, we provide initial research in activity recognition based on a bar scenario, which is the main demonstration scenario of the JAMES project. A robot bartender serves humans and should be able to provide appropriate human-robot interaction in a socially acceptable way. Our task shall be to recognize activities of humans at a bar, for example if they are drinking or requesting the attention of the bartender. According to these states, the robot can interact with them. For example, the robot can avoid talking to a drinking person, or when he is clinking glasses, the robot can respond with "Cheers!". These are interactions which might help the robot to behave in a more social way, so that humans might feel more comfortable interacting with him and accepting his role as a bartender.

Another motivation for activity recognition is to study the machine learning based approach to solve this task. As we will discuss in the next section, we make use of hidden Markov models (HMMs) and hence are interested in their resulting performance on this problem. In combination with the Kinect, a motion sensor device providing joint data of tracked persons, we will try to learn an activity by examples.

1.2 Related Work

Let us now take a look at related work concerning our activity recognition task.

At first, Sotzek [26] and Lenz et al. [17] present a workflow recognition for complex assembly tasks for the hybrid assembly demonstration platform JAHIR [2] (Joint-Action for Humans and Industrial Robots). Thereby, to detect different movements both arm, left-to-right continuous hidden Markov models (HMMs) are used. Evaluation of the results reveals that this is a convincing solution and robust to several changes in the environment, providing an accuracy value of about 90%. It is shown that HMMs are able to compensate different speed and a different trajectory of the arm movement. Moreover, Sotzek presents a solid foundation for the use of the Hidden Markov Model Toolkit (HTK) with non-speech related data. These results reveal that hidden Markov models might be a very good choice for our activity recognition task.

Whereas Ji and Liu [15] present an HMM based human action recognition based on image sequences resulting in a reasonable recognition tool, even Yamato et al. [32] in 1992 applied HMMs to images of sport scenes resulting in a recognition rate of more than 90%.

Berndt and Dietmayer [4, 5] present an intention recognition of human drivers in a car, based on HMMs and several environment observation signals. According to Berndt and Dietmayer, initial tests showed promising recognition results, but further work is needed to improve and adjust this approach. Hasan' et al. [10] and Sanchez et al. [25] also apply HMMs for activity recognition in their studies. Moreover, Wang et al. [30] uses haptic data and HMMs to estimate the human intention to move his or her arm. However, this emphasizes that HMMs should be a reasonable technique for our activity recognition task.

Kerstin Huth [13] investigates characteristics and behaviors of humans in a bar scenario by evaluating videos from real bars. Inspired by the results of her work, we can later derive some reasonable and useful activities for our bar scenario. For example, Huth shows that guests most likely do not use gestures to request attention. Moreover, the body alignment and, to emphasize attention requests, leaning forward towards the bartender are strong social signals. Thus, we can make use of the skeleton data to differentiate between those activities.

1.3 Outline

Having discussed the motivation for applying hidden Markov models to our problem, in the second (next) chapter, we will review the theory of hidden Markov models (HMMs).

Starting with the formalization of Markov models, we extend them and introduce the hidden part. Focusing on a clear and comprehensible mathematical derivation, we will discuss the most important algorithms for HMMs with the goal to actually train our model to learn from training data. At the end, we extend our HMM structure further to pave the way for us to implement them in our context.

In the third chapter, we will discuss our scenario, which is based on human-robot interaction at a bar. First, we introduce the JAMES environment we are working at, then we deliver a detailed description of our bar scenario including possible activities and transitions between them.

The actual implementation part will be presented in the fourth chapter. There, we introduce the hard- and software tools helping us to put the theory discussed so far into practice. In the process, we especially focus on the usage of our software program "Kinactivity", which we will use to record data, train and test our HMM.

In the final and fifth chapter we first evaluate the results given by different HMMs, in order to find the best HMM structure for our data. We then analyze this HMM in detail and discuss the results in the context of our bar scenario. Finally, we summarize our work and briefly discuss future work.

2 Hidden Markov Models

In order to gain a deeper understanding of most processes in our world, one has to consider the relationship between the observed states of that process over time. Whereas we could also treat every data we observe as independent and identically distributed (i.i.d), we will now take a look at a model which discards that constraint and hence is designed to describe *sequential data*.

Our first approach to deal with sequential data will introduce Markov models. On this basis, we will then discuss in detail the extension to hidden Markov models. After formalizing them, we can finally turn to solving fundamental problems with the focus on the learning part of our model. To conclude, we finally consider continuous multidimensional models, as up until then we only assume discrete one-dimensional models. With this knowledge we can then apply hidden Markov models to our activity recognition task in the upcoming chapters.

Note that Rabiner [23] and Bishop [6] serve as the scientific basis of this chapter.

2.1 Introduction to Markov models

Sequential data can be of any kind, for instance measurement of time series, nucleotide base pairs along a strand of DNA or the sequence of characters in an english sentence, as stated in Bishop [6, p. 605], but in most cases we consider temporal sequences, as we will do in this chapter.

In opposition to deterministic models, where we exploit known specific properties of the process, we will deal with statistical (and thereby non-deterministic) models, in which we try to characterize only the statistical properties of the process, so does Rabiner [23] distinguish between these models.

Let us further assume that we view the process as a series of snapshots (discrete time slices)[1] and that at each time t, we can describe the current state of the process by a single discrete random variable X_t. The values of this variable are the states $\mathbf{S} = \{S_1, ..., S_N\}$ of the N-state model.

[1]We assume the interval between time slices is fixed, so we can label times by integers, as pointed out by Russell and Norvig [24].

First of all, we can express the joint distribution for a sequence of T observations using the product rule by

$$p(\mathbf{X}) = p(X_1) \prod_{t=2}^{T} p(X_t | X_1, ..., X_{t-1}) \tag{2.1}$$

where $\mathbf{X} = \{X_1, ..., X_T\}$ denotes the set of the T random variables with $X_i \in \mathbf{S},\ i \in \{1, ..., T\}$.

Let us consider a small example given by Bishop [6]. Imagine, we have a time series of recent observations of binary variables denoting whether on a particular day it rained or not. We wish to predict whether it will rain on the next day. In order to solve this and especially any other more complex problem utilizing (2.1), in practice, we need to make two more important assumptions.

First of all, from now on, we assume our (stochastic) process to be *stationary*, so that the joint probability distribution (and with it the statistical properties of that process [23]) does not vary over time. In other words, as stated by Bishop [6], the conditional distributions $p(X_t | X_1, ..., X_{t-1})$ from which the states evolve in time remain the same. A model satisfying this assumption is called a *homogeneous* model.

Secondly, our current state should just be affected by a *finite* set of preceding states. As Bishop [6] explains, future predictions should be independent of all but the most recent observations. This assumption is called *Markovian assumption* and processes satisfying it are called Markov models, named by the russian statistician Andrey Markov[2]. The degree of dependency is given by the order of the model.

From now on, we will just consider first-order (homogeneous) Markov models, in which the current state just depends on the previous one. This leads us to a simplified version of (2.1) given by

$$p(\mathbf{X}) = p(X_1) \prod_{t=2}^{T} p(X_t | X_{t-1}). \tag{2.2}$$

Furthermore, the conditional distributions correspond to a static *state transition matrix* $\mathbf{A}$ of dimension $N \times N$. For any time t, the elements of $\mathbf{A}$ are known as *transition probabilities* and defined by

$$A_{jk} = p(X_t = S_k | X_{t-1} = S_j),\ 1 \leq j, k \leq N \tag{2.3}$$

[2] `http://en.wikipedia.org/wiki/Andrey_Markov`

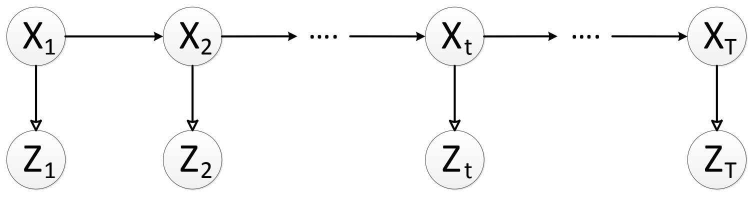


Figure 2.1: **Graphical structure of hidden Markov models in general over the course of time** - Two stochastic processes take place: the hidden state transition (Markov process) at each time slice (upper arrows) and the resulting observation at each state (lower arrows).

where $\sum_{k=1}^{N} A_{jk} = 1$ and $0 \leq A_{jk} \leq 1$ because of standard stochastic constraints. In order to calculate the special case $p(X_1)$ we introduce the initial state probability vector $\boldsymbol{\pi}$ of dimension N. Their elements are defined by

$$\pi_i = p(X_1 = S_i),\ 1 \leq i \leq N \tag{2.4}$$

where again $\sum_{i=1}^{N} \pi_i = 1$ and $0 \leq \pi_i \leq 1$. Our Markov model is hence defined by the model parameter set $\boldsymbol{\lambda} = (\boldsymbol{\pi}, \mathbf{A})$.

2.2 Extension to hidden Markov models

Being more realistic, many processes in real-world can not be observed directly, but we can often observe their signals and their effects on the environment. Strictly speaking, our only way to identify the environment is by using our sensors and senses. It is impossible to directly observe concepts like "rain", we can just observe the effects of "rain" to our senses, for example seeing or hearing raindrops or feeling the wet ground. Let us thereby now differentiate between the random variables concerning the states $\mathbf{X}$ and the observations, denoted by $\mathbf{Z} = \{Z_1, ..., Z_T\}$.

Introducing unobserved (hidden) states leads us to the concept of *hidden Markov models* (HMMs), which are instances of state space models[3]. In contrast to regular Markov models we discussed before, the states $\mathbf{X}$ are not directly observable, but still form the Markov process. Figure 2.1 illustrates the resulting concept of HMMs.

The observable variables $\mathbf{Z}$ can either be discrete or continuous, and a variety of different conditional distributions can be used to model them, as pointed out

[3]Besides HMMs, linear dynamical systems are one of the most important examples of state space models in which the state and observable variables are Gaussians, as described by Bishop [6].

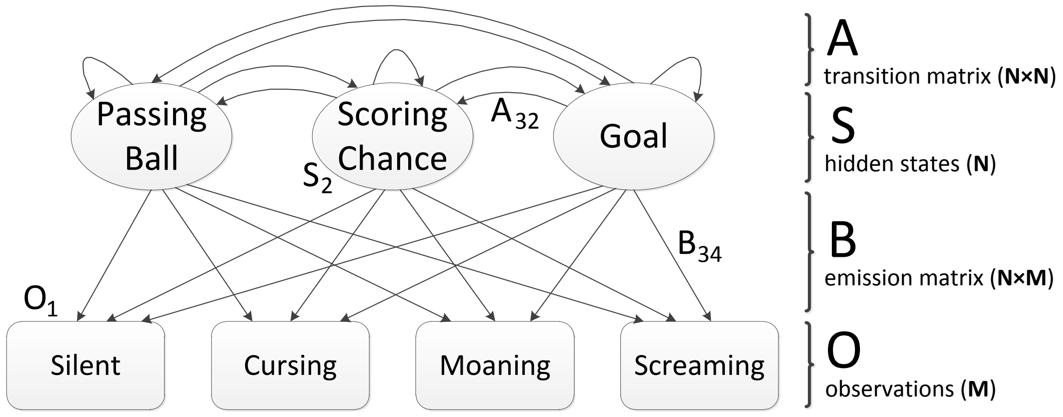


Figure 2.2: **Example of a hidden Markov model (HMM)** - N=3 hidden states and M=4 possible observations (discrete variables) illustrated in a state transition diagram with the corresponding labelling.

by Bishop [6]. Let us in first instance consider discrete observations, so that $Z_i \in \mathbf{O}$, $1 \leq i \leq T$ with $\mathbf{O} = \{O_1, ..., O_M\}$ being a discrete set of M possible observations. We will later discuss the continuous case. However, at any time t, the observable variable Z_t does only depend on the current hidden state of X_t. Vice versa, at each time t the state X_t has a probability distribution over the observable states and hence emmits a randomly picked observation. We can denote this emission distribution by $b_j(k) = p(Z_t = O_k | X_t = S_j)$, $1 \leq j \leq N$, $1 \leq k \leq M$ at any time t, as does Rabiner [23]. Again, because of our homogeneity, the emission distributions for each state do not change over time. Furthermore, in case of discrete observations, we will store them in a single *emission matrix* $\mathbf{B}$ of dimension $N \times M$ with entries $B_{jk} = b_j(k)$ and the standard stochastic constraints $\sum_{k=1}^{N} B_{jk} = 1$ and $0 \leq B_{jk} \leq 1$.

For a sequence of T observations, this leads us to the complete joint distribution over both hidden states and observable variables, as stated by Bishop [6] and Russell and Norvig [24], given by

$$p(\mathbf{X}, \mathbf{Z} | \boldsymbol{\lambda}) = \underbrace{p(X_1 | \boldsymbol{\pi}) \left[\prod_{t=2}^{T} p(X_t | X_{t-1}, \mathbf{A}) \right]}_{\text{transition}} \underbrace{\prod_{t=1}^{T} p(Z_t | X_t, \mathbf{B})}_{\text{emission}} \tag{2.5}$$

in which the set of model parameters $\boldsymbol{\lambda} = \{\boldsymbol{\pi}, \mathbf{A}, \mathbf{B}\}$ completely governs our HMM.

Before we move on, let us now take a brief look at a simple example of a hidden Markov model. Assume the following scenario. You are watching an exciting football match in the middle of a vast crowd of people not being able to see the screen. By listening to the surrounding noises you imagine to infer the ongoing action on the screen. Figure 2.2 shows a possible hidden Markov model for such a task in a state transition diagram. Whereas you can observe a silent, cursing, moaning or screaming crowd, the actual states of the game (e.g. passing ball, scoring chance, goal) are hidden.

2.2.1 Different types of hidden Markov models

To take account for specific properties of the observed process, according to Rabiner [23], we can think of different types of interconnections between states. This is done by imposing constraints on the transition matrix $\mathbf{A}$. In our preceding example (Figure 2.2), every hidden state of our HMM could be reached from every other state within a finite number of time slices. We call this an *ergodic* model.

Let us now shortly discuss *left-to-right* models, another important and popular type of HMMs. With *left-to-right* models, we want to model processes whose properties change over time. Thereby, our model should be designed so that our state index can just increase or stay the same with every new period of time (i.e. states proceed from left to right). Before we formalise left-to-right models, let us take a brief look at an example provided by Bishop [6] of their applications. Assume, our goal is to create a model for on-line character recognition. That means, each character is represented by the trajectory of the pen as a sequence of pen coordinates. Thus, the character recognition properties change over time, because the trajectory is a function of time. For example, the digit '2' starts at the top left with a sweeping arc down to the cusp (or loop) at the bottom left, followed by a second more-or-less straight sweep ending at the bottom right, as stated in [6]. It should be clear that the model can not cope with writing the character in reverse order. Another huge application field for left-to-right models is speech recognition. Modelling each word as an own hidden Markov model, we use left-to-right models to represent the acoustic development of that world, which of course changes over time.

Left-to-right models are mathematically described as follows. We first start in state S_1, so that $\pi_1 = 1$. Our model is then constrained by $A_{ij} = 0,\ j < i$ with $1 \leq i, j \leq N$, so that it can just proceed from left to right. Furthermore, we can think of an additional constraint to prevent large state changes (jumps) of more than Δ states. In that case, our preceding constraint on $\mathbf{A}$ would be supplemented by $A_{ij} = 0,\ j > i + \Delta$. To conclude with an example, the following state transition matrix corresponds to a 4-state left-to-right HMM with no more than two jumps (i.e. $\Delta = 2$):

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & 0 \\ 0 & A_{22} & A_{23} & A_{24} \\ 0 & 0 & A_{33} & A_{34} \\ 0 & 0 & 0 & A_{44} \end{pmatrix} \tag{2.6}$$

2.3 Solving fundamental problems for hidden Markov models

With respect to Rabiner [23], let us now address three fundamental problems that should be solved in order to apply our HMM to real-world applications:

1. **Evaluation:** How can we calculate the probability of a sequence of observations generated by a specific HMM?

2. **Decoding:** How can we determine the most likely sequence of hidden states of a specific HMM given a sequence of observations?

3. **Learning:** How can we adjust the model parameters of a specific HMM so that it generates a sequence of observations with maximum likelihood?

Note that Leonard E. Baum et al. [20, 19, 18, 3] and Andrew Viterbi [29] contributed essentially to solving these problems.

2.3.1 Evaluation and the forward-backward algorithm

The evaluation problem is based upon the task of calculating $p(\mathbf{Z}|\boldsymbol{\lambda})$. Given the observation sequence (stored in $\mathbf{Z}$) and our HMM (defined by $\boldsymbol{\lambda}$) we want to evaluate how well that model matches that observation sequence. Evaluation is needed for choosing correctly among several competing models for that one that best matches the observations. For example, a simple isolated word speech recognizer based on HMMs recognizes an unknown word (i.e. the observation sequence, which consists of speech vectors) by evaluating each word-model and choosing the word whose model has the highest likelihood of generating the unknown word.

A first naive attempt to solve the evaluation problem can be done based on (2.5) by marginalizing over all possible state sequences giving

$$\begin{aligned} p(\mathbf{Z}|\boldsymbol{\lambda}) &= \sum_{\text{all } \mathbf{X}} p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\lambda}) && (2.7) \\ &= \sum_{\text{all } \mathbf{X}} \left[p(X_1|\boldsymbol{\pi}) \left[\prod_{t=2}^{T} p(X_t|X_{t-1}, \mathbf{A}) \right] \prod_{t=1}^{T} p(Z_t|X_t, \mathbf{B}) \right]. && (2.8) \end{aligned}$$

Solving term on the right is highly inefficient, because we have N reachable states for each time t, which results in N^T possible state sequences. Thus, the calculation grows exponentially with the lenght of the observation sequence and results in computational cost that scale like $O(N^T)$.

Around the year 1970, Leonard E. Baum et al. [20, 19, 18, 3] laid the foundation for a much more efficient way to solve the evaluation problem which is known as the *forward-backward algorithm* and uses a recursive approach. At first, let us define the forward variable $\alpha_i(X_t)$ as

$$\alpha_i(X_t) = p(Z_1, ..., Z_t, X_t = S_i|\boldsymbol{\lambda}), \quad 1 \leq i \leq N, \ 1 \leq t \leq T \qquad (2.9)$$

which represents the joint probability of the partial observation from start up to time t *and* that our model is in state S_i at time t. Then we can express the desired calculation of $p(\mathbf{Z}|\boldsymbol{\lambda})$ by marginalizing over all possible end-states at time $t = T$ as follows:

$$p(\mathbf{Z}|\boldsymbol{\lambda}) = \sum_{i=1}^{N} \alpha_i(X_T). \qquad (2.10)$$

Not let us take a look at how to calculate the forward variables $\alpha_i(X_t)$. In line with Bishop [6, p. 620] we will first derive the terms recursively by means of basic conditional independence properties, and then illustrate the solution with help of a HMM lattice diagram as presented by Rabiner [23].

At first, in order to start our recursive approach, we calculate the initial term given by

$$\begin{aligned} \alpha_i(X_1) &= p(Z_1, X_1 = S_i|\boldsymbol{\lambda}) && && (2.11) \\ &= \pi_i \cdot p(Z_1 = O_k|X_1 = S_i, \boldsymbol{\lambda}) && 1 \leq k \leq M && (2.12) \\ &= \pi_i \cdot b_i(k) && 1 \leq k \leq M. && (2.13) \end{aligned}$$

In the next step we will derive the recursive relation between the forward variables. To keep the derivation clear and short we reduce the expression $X_t = S_i$ and $X_{t-1} = S_j$, respectively, to X_t and X_{t-1} instead, which should be clear in the context:

$$\begin{aligned}
\alpha_i(X_t) &= p(Z_1, ..., Z_t, X_t) \\
&= p(Z_1, ..., Z_t | X_t) \cdot p(X_t) \\
&= p(Z_t | X_t) \cdot p(Z_1, ..., Z_{t-1} | X_t) \cdot p(X_t) \\
&= p(Z_t | X_t) \cdot p(Z_1, ..., Z_{t-1}, X_t) \\
&= p(Z_t | X_t) \sum_{\text{all } X_{t-1}} p(Z_1, ..., Z_{t-1}, X_{t-1}, X_t) \\
&= p(Z_t | X_t) \sum_{\text{all } X_{t-1}} p(Z_1, ..., Z_{t-1}, X_t | X_{t-1}) \cdot p(X_{t-1}) \\
&= p(Z_t | X_t) \sum_{\text{all } X_{t-1}} p(Z_1, ..., Z_{t-1} | X_{t-1}) \cdot p(X_t | X_{t-1}) \cdot p(X_{t-1}) \\
&= p(Z_t | X_t) \sum_{\text{all } X_{t-1}} p(Z_1, ..., Z_{t-1}, X_{t-1}) \cdot p(X_t | X_{t-1}) && (2.14) \\
&= p(Z_t | X_t) \sum_{j=1}^{N} \alpha_j(X_{t-1}) \cdot p(X_t | X_{t-1}) && (2.15) \\
&= b_i(k) \sum_{j=1}^{N} \alpha_j(X_{t-1}) \cdot A_{ji}. && (2.16)
\end{aligned}$$

Up to incl. (2.14) we made use of basic conditional independence properties written down in Bishop [6, p. 619]. We then obtain the recursive core in (2.15) by applying definition (2.9). To conclude, we express this solution by using our basic model parameters and the definition $Z_t = O_k,\ 1 \leq k \leq M$ which leads to (2.16).

The left lattice diagram in figure 2.3 illustrates the sum-term of the forward recursion solution (2.16): If we consider (i.e. sum up) the α_j's of each state in the previous time segment weighted by the transition probability A_{ji}, which results in state S_i, we almost obtain α_i for the current time segment. All that is left is to account for the new observation (in (2.16 called O_k) which leads to multiplying the previous result with $b_i(k)$ and we are done.

In contrast to our first naive attempt (2.8) which led to computational cost that scale like $O(N^T)$ we now just have overall costs of $O(N^2)$: At each time t (there are T time segments) for each state (there are N states) we have to compute the N terms on the right of (2.16) which results in TN^2 calculations. This refinement

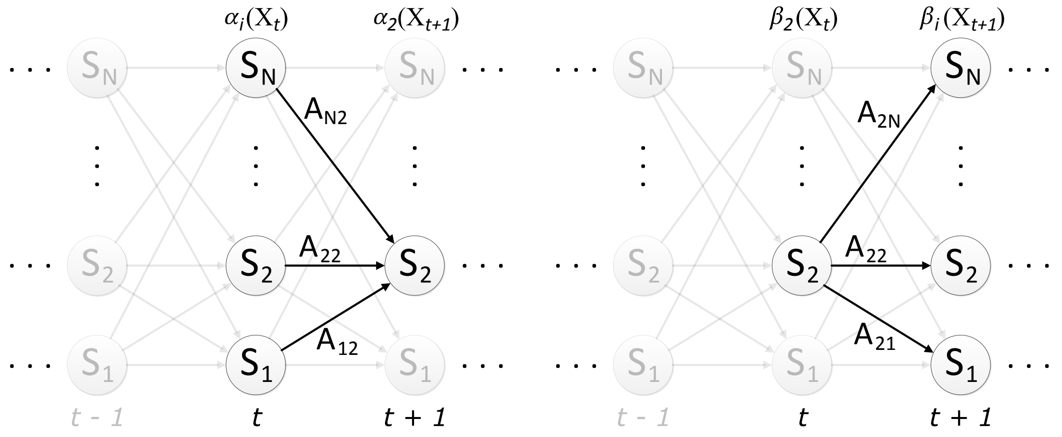


Figure 2.3: **Lattice (or trellis) diagrams for N-state HMMs over time** - *Left:* Illustration how the forward variable $\alpha_2(X_{t+1})$ is related to its predecessors $\alpha_i(X_t)$. *Right:* Illustration how the backward variable $\beta_2(X_t)$ is related to its successors $\beta_i(X_{t+1})$.

really plays a huge role in order to scale our algorithm with the length of the observation sequence.

Let us now introduce the counterpart of the forward variables $\alpha_i(X_t)$, these are the backward variables $\beta_i(X_t)$. Although we do not need them solving the problem of evaluation which we have already done with our forward variables, they will be used to help solve fundamental upcoming problems concerning decoding and learning. The backward variable $\beta_i(X_t)$ is defined by

$$\beta_i(X_t) = p(Z_{t+1}, ..., Z_T | X_t = S_i, \boldsymbol{\lambda}), \quad 1 \leq i \leq N, \ 1 \leq t \leq T \tag{2.17}$$

which represents the joint probability of the partial observation from time $t+1$ up to the end given that our model is in state S_i at time t. Note that we will define a backward algorithm (used for message passing as we will discuss later) so that our recursion will start at $t = T$ with the initial term (necessarily) defined as

$$\beta_i(X_T) = 1, \quad 1 \leq i \leq N. \tag{2.18}$$

We will skip the complete derivation of the recursive relation between the backward variables as it is quite similar to the derivation of the forward variables:

$$\beta_i(X_t) = \sum_{j=1}^{N} \beta_j(X_{t+1}) \cdot p(X_{t+1}|X_t) \cdot p(Z_{t+1}|X_{t+1}) \tag{2.19}$$

$$= \sum_{j=1}^{N} \beta_j(X_{t+1}) \cdot A_{ij} \cdot b_j(l). \tag{2.20}$$

Again this result can be well illustrated by the right lattice diagram in figure 2.3 showing the first two terms of the marginalization step of (2.19). In addition, we need to account for the observation of each successor which leads to the term $p(Z_{t+1}|X_{t+1})$. One can easily verify that this result is in line with definition (2.17). Finally, applying our basic model parameters and the definition $Z_{t+1} = O_l,\ 1 \leq l \leq M$ leads to (2.20).

2.3.2 Decoding and the Viterbi algorithm

Our next problem is about decoding and deals with uncovering the hidden part of an HMM. Given the observation sequence (stored in $\mathbf{Z}$) and our HMM (defined by $\boldsymbol{\lambda}$) we want to decode the state sequence (stored in $\mathbf{X}$) which explains our observations with maximum likelihood. Solving this problem will help us gaining a better understanding of the model structure and especially the (physical) meaning of the model states in order to adjust our model accordingly in the next step. Furthermore, for example in the context of speech recognition, decoding can be used to find the most probable phoneme sequence given the acoustic signal observations.

First of all we should clarify that we are not interested in searching the set of states that are *individually* the most probable, as it does not consider the state transitions. Two successive states that are individually the most probable can represent a sequence having zero probability if the state transition probability between them is zero. That is why we are interested in finding the *single* most probable state *sequence*, i.e. in maximizing $p(\mathbf{X}|\mathbf{Z}, \boldsymbol{\lambda})$ which is equivalent to $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\lambda})$. An efficient way to calculate $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\lambda})$ is given by the so called Viterbi algorithm.

The Viterbi algorithm was first described by Andrew Viterbi in 1967 [29] in the context of decoding convolutional codes and was further reviewed by Forney [7] in 1973. This algorithm belongs to the class of dynamic programming algorithms and hence solves our problem by breaking it down into simpler subproblems. As a result of having a first-order hidden Markov model, our most likely sequence of hidden states up to time t depends only on the most likely sequences up to time

$t-1$, the observation at time t and the state transition probabilities. For each state at each time t we only need to keep track of the most likely path that leads to that state (survivor path). But before going into detail, let us start by defining the probability quantities

$$\tilde{\delta}(X_t) = \max_{X_1,...,X_{t-1}} p(X_1, ..., X_t, Z_1, ..., Z_t|\boldsymbol{\lambda}) \tag{2.21}$$

$$\delta(X_t) = \max_{X_1,...,X_{t-1}} \ln p(X_1, ..., X_t, Z_1, ..., Z_t|\boldsymbol{\lambda}). \tag{2.22}$$

$\tilde{\delta}(X_t)$ (2.21) stores the probability of the most likely path of states which ends in the defined state X_t and accounts for the first t observations. Whereas $\tilde{\delta}$ uses floating-point arithmetic, δ defined by (2.22) uses log probabilities throughout the computations in order to cope with the problem of underflow in the results.

In the same manner as done with the forward variables, we will define $\tilde{\delta}$ and δ recursively starting with $t = 1$ which leads to

$$\tilde{\delta}(X_1) = p(X_1, Z_1|\boldsymbol{\lambda}) \tag{2.23}$$

$$= \pi_i \cdot b_i(k) \tag{2.24}$$

$$\delta(X_1) = \ln \pi_i + \ln b_i(k). \tag{2.25}$$

By induction similar to the derivation of (2.15) we obtain

$$\tilde{\delta}(X_t) = p(Z_t|X_t) \cdot \max_{X_{t-1}} \left[\tilde{\delta}(X_{t-1}) \cdot p(X_t|X_{t-1})\right] \tag{2.26}$$

$$= b_i(k) \cdot \max_{X_{t-1}} \left[\tilde{\delta}(X_{t-1}) \cdot A_{ji}\right] \tag{2.27}$$

$$\delta(X_t) = \ln b_i(k) + \max_{X_{t-1}} [\delta(X_{t-1}) + \ln A_{ji}] \tag{2.28}$$

where $2 \leq t \leq T$ and again $X_t = S_i, X_{t-1} = S_j$ for defined i, j with $1 \leq i, j \leq N$ and $Z_t = O_k$ for a defined k with $1 \leq k \leq M$. Note that, in contrast to the forward variables, we maximize over previous states X_{t-1} instead of summing.

Figure 2.4 illustrates the Viterbi algorithm, especially the results of (2.27), using some random numbers for the model parameters and the resulting probability of the most likely path for each state in time.

To conclude, we can calculate the desired probability $p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\lambda})$ of the final most likely path up to time T (called the Viterbi path) as follows:

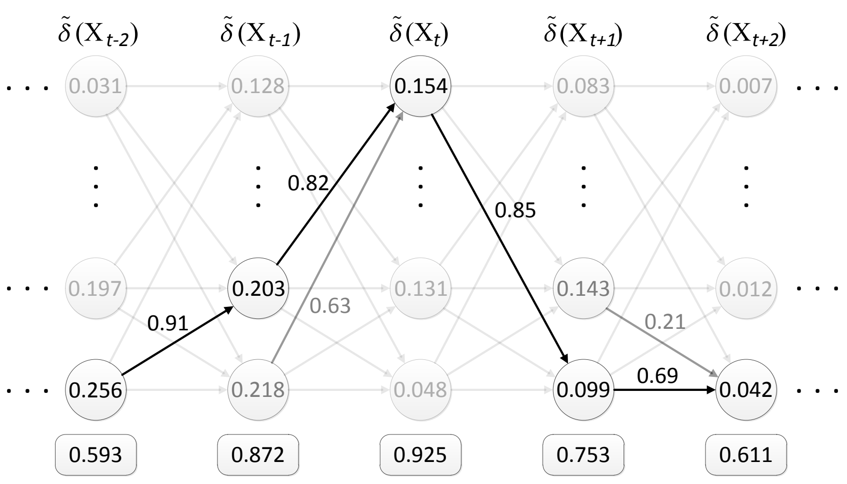


Figure 2.4: **HMM lattice (or trellis) diagram illustrating the Viterbi algorithm** - To efficiently determine the probability of the most likely path at time t for a defined state $X_t = S_i$, the Viterbi algorithm looks for any state $X_{t-1} = S_j$ at time $t-1$ and chooses the one that has the highest probability of its most likely path $\tilde{\delta}(X_{t-1})$ weighted by the transition probability A_{ji}. To obtain the resulting probability $\tilde{\delta}(X_t)$ we also have to account for the observation at time t whose probability is given by the rounded rectangle at the bottom.

$$p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\lambda}) = \max_{X_T} \ \tilde{\delta}(X_T) \tag{2.29}$$

$$= \exp\left(\max_{X_T} \ \delta(X_T)\right) \tag{2.30}$$

In order to retrieve the corresponding state sequence to the Viterbi path we just need to keep track of the actual states that are chosen by the max-terms. Therefore, Rabiner [23] and Bishop [6] introduce a function $\psi(X_t)$ which stores for each defined state X_t the last state of its most likely state sequence and is computed along with (2.27), resp. (2.28). Once we reached the end $t = T$ and hence found the Viterbi path we can use backtracking utilizing $\psi(X_t)$ to easily obtain the resulting state sequence starting at the final state given by $\arg\max_{X_T} \ \tilde{\delta}(X_T)$, resp. $\arg\max_{X_T} \ \delta(X_T)$.

2.3.3 Learning and the EM algorithm

In the third and final section of solving fundamental problems for HMMs we want to discuss the learning or training of our model. Our goal will be to optimally adapt the model parameters $\boldsymbol{\lambda}$ in order to maximize the likelihood function $p(\mathbf{Z}|\boldsymbol{\lambda})$, the probability of our model to generate the given observation (training) sequence stored in $\mathbf{Z}$. This procedure is the key step to create an HMM for our observations, which will often be extracted out of real phenomena. For example in the context of speech recognition, given acoustic signal observations, learning will provide us the ability to optimally estimate model parameters for each word model.

Unfortunately, direct maximization of the likelihood function $p(\mathbf{Z}|\boldsymbol{\lambda})$ is not applicable and there is no known way to optimally adjust the model parameters. With respect to Rabiner [23] and especially Bishop [6, p. 439-441,615-618], we will apply an expectation-maximization (EM) algorithm to locally maximize our likelihood function using an efficient iterative procedure. Note that A. P. Dempster, N. M. Laird, D. B. Rubin [27] (1977) were one of the first presenting this general approach for incomplete data models, such as HMMs. In general, our EM algorithm will work as follows: Starting with an initial selection for the model parameters $\boldsymbol{\lambda}^{old}$, we will reestimate them giving $\boldsymbol{\lambda}^{new}$ by iteratively applying an expectation (E) step followed by a maximization (M) step. At each new iteration we start with our model parameters gained in the previous iteration ($\boldsymbol{\lambda}^{old} \leftarrow \boldsymbol{\lambda}^{new}$). We will now explain this procedure in detail.

Starting with the expectation (E) step, we have to evaluate the posterior distribution of the hidden variables $p(\mathbf{X}|\mathbf{Z}, \boldsymbol{\lambda}^{old})$. To achieve this, let us define the joint and the marginal posterior distribution, denoted as $\xi(X_t, X_{t+1})$ and $\gamma(X_t)$:

$$\xi(X_t, X_{t+1}) = p(X_t, X_{t+1}|\mathbf{Z}, \boldsymbol{\lambda}) \tag{2.31}$$

$$\gamma(X_t) = p(X_t|\mathbf{Z}, \boldsymbol{\lambda}) \tag{2.32}$$

$$= \sum_{X_{t+1}} \xi(X_t, X_{t+1}). \tag{2.33}$$

Given the observation sequence $\mathbf{Z}$ and the model parameters $\boldsymbol{\lambda}$, $\xi(X_t, X_{t+1})$ (2.31) is defined as the joint event probability of being in states specified by the two successive latent variables X_t and X_{t+1}, whereas $\gamma(X_t)$ (2.32) represents the probability of just being in state specified by the latent variable X_t. Furthermore, γ can be expressed in terms of ξ by marginalizing over X_{t+1} as denoted in (2.33).

With the results of the forward-backward algorithm we can now evaluate $\gamma(X_t)$ and $\xi(X_t, X_{t+1})$. Let us start with $\gamma(X_t)$:

$$\gamma(X_t) = \frac{p(\mathbf{Z}|X_t)\cdot p(X_t)}{p(\mathbf{Z})} = \frac{p(\mathbf{Z}|X_t)\cdot p(X_t)}{\sum\limits_{X_t}[p(\mathbf{Z}|X_t)\cdot p(X_t)]} \tag{2.34}$$

$$= \frac{p(Z_1,...,Z_t,X_t)\cdot p(Z_{t+1},...,Z_T|X_t)}{p(\mathbf{Z})} \tag{2.35}$$

$$= \frac{\alpha(X_t)\cdot\beta(X_t)}{p(\mathbf{Z})} = \frac{\alpha(X_t)\cdot\beta(X_t)}{\sum\limits_{X_t}[\alpha(X_t)\cdot\beta(X_t)]}. \tag{2.36}$$

After using Bayes' theorem in (2.34) and the conditional independence property in (2.35), we can finally substitute the existing terms utilizing our forward and backward variables $\alpha(X_t)$ and $\beta(X_t)$ which leads us to (2.36). The same procedure can be applied for $\xi(X_t, X_{t+1})$ giving

$$\xi(X_t,X_{t+1}) = \frac{p(\mathbf{Z}|X_t,X_{t+1})\cdot p(X_t,X_{t+1})}{p(\mathbf{Z})} \tag{2.37}$$

$$= \frac{p(Z_1,...,Z_t|X_t)p(Z_{t+1}|X_{t+1})p(Z_{t+2},...,Z_T|X_{t+1})p(X_{t+1}|X_t)p(X_t)}{p(\mathbf{Z})} \tag{2.38}$$

$$= \frac{\alpha(X_t)\cdot p(Z_{t+1}|X_{t+1})\cdot p(X_{t+1}|X_t)\cdot\beta(X_{t+1})}{p(\mathbf{Z})} \tag{2.39}$$

$$= \frac{\alpha(X_t)\cdot b_j(k)\cdot A_{ij}\cdot\beta(X_{t+1})}{p(\mathbf{Z})} \tag{2.40}$$

where again $X_t = S_i$, $X_{t+1} = S_j$ and $Z_{t+1} = O_k$ for defined i,j,k with $1 \leq i,j \leq N$ and $1 \leq k \leq M$.

After evaluating the joint and the marginal posterior distribution and thereby the general posterior distribution of the hidden variables $p(\mathbf{X}|\mathbf{Z},\boldsymbol{\lambda}^{old})$, we will now apply this result to calculate the expected value of the complete-data log likelihood $p(\mathbf{X},\mathbf{Z}|\boldsymbol{\lambda})$. This function, denoted as $\mathcal{Q}(\boldsymbol{\lambda},\boldsymbol{\lambda}^{old})$, is evaluated for the general parameter value $\boldsymbol{\lambda}$, given the current estimate of the model parameters $\boldsymbol{\lambda}^{old}$, and defined as follows:

$$\mathcal{Q}(\boldsymbol{\lambda},\boldsymbol{\lambda}^{old}) = \sum_{\mathbf{X}} p(\mathbf{X}|\mathbf{Z},\boldsymbol{\lambda}^{old})\cdot\ln p(\mathbf{X},\mathbf{Z}|\boldsymbol{\lambda}) \tag{2.41}$$

$$= \sum_{i=1}^{N}[\gamma(X_1 = S_i)\cdot\ln\pi_i] + \sum_{t=1}^{T-1}\sum_{i=1}^{N}\sum_{j=1}^{N}[\xi(X_t = S_i, X_{t+1} = S_j)\cdot\ln A_{ij}]$$

$$+ \sum_{t=1}^{T}\sum_{i=1}^{N}[\gamma(X_t = S_i)\cdot\ln b_i(k)]\,. \tag{2.42}$$

Following Bishop [6, p. 616-617], to obtain equation (2.42), we make use of our results for the joint and the marginal posterior distribution and (2.5). Again, note that in the last term in (2.42), we assume $Z_t = O_k$ for a defined k with $1 \leq k \leq M$.

Let us now turn to the maximization (M) step, where we maximize $\mathcal{Q}(\boldsymbol{\lambda}, \boldsymbol{\lambda}^{old})$ with respect to $\boldsymbol{\lambda}$ and evaluate $\boldsymbol{\lambda}^{new}$ given by

$$\boldsymbol{\lambda}^{new} = \arg\max_{\boldsymbol{\lambda}} \mathcal{Q}(\boldsymbol{\lambda}, \boldsymbol{\lambda}^{old}). \tag{2.43}$$

Using appropriate Lagrange multipliers for the maximization part in (2.43), as suggested in Bishop [6, p. 617] and Rabiner [23], this leads us to the following final set of reestimation formulas:

$$\pi_i^{new} = \frac{\gamma(X_1 = S_i)}{\sum\limits_{X_1} \gamma(X_1)} = \gamma(X_1 = S_i) \tag{2.44}$$

$$A_{ij}^{new} = \frac{\sum\limits_{t=1}^{T-1} \xi(X_t = S_i, X_{t+1} = S_j)}{\sum\limits_{k=1}^{N} \sum\limits_{t=1}^{T-1} \xi(X_t = S_i, X_{t+1} = S_k)} = \frac{\sum\limits_{t=1}^{T-1} \xi(X_t = S_i, X_{t+1} = S_j)}{\sum\limits_{t=1}^{T-1} \gamma(X_t = S_i)} \tag{2.45}$$

$$b_i^{new}(k) = \frac{\sum\limits_{t=1}^{T} \begin{cases} \gamma(X_t = S_i), & Z_t = O_k \\ 0, & Z_t \neq O_k \end{cases}}{\sum\limits_{t=1}^{T} \gamma(X_t = S_i)}. \tag{2.46}$$

According to Rabiner [23], let us take a brief moment to understand why these results are reasonable. For example, in case of the transition probability A_{ij}^{new} (2.45), the numerator can be interpreted as the expected number of transitions from state S_i to S_j, whereas the denominator represents the expected number of times that S_i is even visited. Furthermore, the emission probability $b_i^{new}(k)$ (2.46) is updated according to the ratio between the expected number of times being in state S_i *and* observing O_k, and being in state S_i at all.

To conclude, we will shortly recall the procedure to train a hidden Markov model using the EM algorithm. Starting with an initial (valid) selection of the model parameters $\boldsymbol{\lambda}^{old}$, we first apply the forward-backward algorithm to evaluate the α and β terms. We then use the results to approach the E step, i.e. evaluate both $\gamma(X_t)$ and $\xi(X_t, X_{t+1})$ and hence $\mathcal{Q}(\boldsymbol{\lambda}, \boldsymbol{\lambda}^{old})$. Finally, the M step gives us revised model parameters $\boldsymbol{\lambda}^{new}$. We then continue applying the E and M step using our new model parameters ($\boldsymbol{\lambda}^{old} \leftarrow \boldsymbol{\lambda}^{new}$) until some convergence criterion is satisfied, for instance $\left|p(\mathbf{Z}|\boldsymbol{\lambda}^{new}) - p(\mathbf{Z}|\boldsymbol{\lambda}^{old})\right| < t$, i.e. the difference between old and new evaluation of our likelihood function is below some threshold t. Luckily for us, as stated in Bishop [6, p. 440] and Rabiner [23], the general EM algorithm has the property that each iteration will never decrease $p(\mathbf{Z}|\boldsymbol{\lambda})$, i.e. $p(\mathbf{Z}|\boldsymbol{\lambda}^{new}) \geq p(\mathbf{Z}|\boldsymbol{\lambda}^{old})$, unless we reached a local maximum, but there we finish our training procedure.

2.4 Extension to continuous multidimensional HMMs

To this point, we just considered 1-dimensional discrete observations for our hidden Markov model. In practice, however, we are often dealing with continuous and/or multidimensional data, as we will later when applying the theory of this chapter to our bar scenario having a vector of multiple continuous joint positions. So let us now briefly discuss the case of $\mathbf{O}$ being a D-dimensional vector of continuous values and $\boldsymbol{Z_t} \in \mathbf{O}$, $1 \leq t \leq T$, with $\boldsymbol{Z_t}$ in bold characters as it now represents a vector.

In order to define our new continuous emission distribution $b_j(\boldsymbol{Z_t})$, we will apply a mixture of K multivariate Gaussians, as they represent a widely used model for the distribution of continuous variables. Thus, the emission distribution takes the form

$$b_j(\boldsymbol{Z_t}) = \sum_{k=1}^{K} c_{jk} \cdot \mathcal{N}(\boldsymbol{Z_t}|\boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}), \qquad 1 \leq j \leq N, \quad 1 \leq t \leq T \tag{2.47}$$

where c_{jk} denotes the mixing coefficient, $\boldsymbol{\mu}_{jk}$ the D-dimensional mean vector, and $\boldsymbol{\Sigma}_{jk}$ the $D \times D$ covariance matrix for the kth mixture component in state j. To obtain a valid emission distribution, the stochastic constraints $\sum_{k=1}^{K} c_{jk} = 1$ and $c_{jk} \geq 0$ for $1 \leq j \leq N, 1 \leq k \leq K$ should be satisfied.

Now, the set of model parameters which completely governs our HMM is described as $\boldsymbol{\lambda} = \{\boldsymbol{\pi}, \mathbf{A}, \mathbf{c}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. Hence we need new reestimation formulas for the parameters of the Gaussian mixture model. Reviewing the EM algorithm discussed before, the E step remains the same, except for the emission distribution in (2.42), where we make use of definition (2.47). Moreover, maximizing (2.42) in the M step with respect to our Gaussian mixture model parameters just needs to consider the final term, so that we can independently apply EM for Gaussian mixtures, as discussed by Bishop [6, p. 435-443], which results in

$$c_{jk}^{new} = \frac{\sum_{t=1}^{T} \gamma(X_t = S_j, k)}{\sum_{t=1}^{T} \sum_{k=1}^{K} \gamma(X_t = S_j, k)} \tag{2.48}$$

$$\boldsymbol{\mu}_{jk}^{new} = \frac{\sum_{t=1}^{T} \gamma(X_t = S_j, k) \cdot \boldsymbol{Z_t}}{\sum_{t=1}^{T} \gamma(X_t = S_j, k)} \tag{2.49}$$

$$\boldsymbol{\Sigma}_{jk}^{new} = \frac{\sum_{t=1}^{T} \gamma(X_t = S_j, k) \cdot (\boldsymbol{Z_t} - \boldsymbol{\mu}_{jk}^{new})(\boldsymbol{Z_t} - \boldsymbol{\mu}_{jk}^{new})^T}{\sum_{t=1}^{T} \gamma(X_t = S_j, k)}. \tag{2.50}$$

The linking between EM for Gaussian mixtures and EM as discussed before is reflected in the new marginal posterior distribution $\gamma(X_t = S_j, k)$, where we consider, given the observation Z_t, the probability of being in state j *and* the probability of mixture component k being responsible for the observation. Thus, we need to multiply both so called responsibilities resulting in

$$\gamma(X_t = S_j, k) = \gamma(X_t) \cdot \frac{c_{jk} \cdot \mathcal{N}(\boldsymbol{Z_t} | \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})}{\sum_{k=1}^{K} c_{jk} \cdot \mathcal{N}(\boldsymbol{Z_t} | \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})} \tag{2.51}$$

where $\gamma(X_t)$ is evaluated as previously using equation (2.36). Note that, similar to the reestimation formulas (2.44)-(2.46), our new ones (2.48)-(2.50) can also be verified easily by thinking in terms of expectation values, as suggested by Rabiner [23]. For example, c_{jk}^{new} represents the ratio between the expected number of times our model is in state j *and* using the kth mixture component, and the expected number of times the system is in state j at all. The same thoughts can be applied to verify (2.49) and (2.50).

3 Scenario

In this chapter, we will discuss our human-robot interaction scenario and derive a model which can then be implemented in the later stages. Let us first introduce the JAMES project (Joint Action for Multimodal Embodied Social Systems).

3.1 JAMES environment

The JAMES project (Joint Action for Multimodal Embodied Social Systems) [28, 8] researches the area of robots having a socially appropriate behavior and interact in that way with humans. The main demonstration scenario of the JAMES project is based upon a bar scenario in which the robot, shown in figure 3.1, serves human guests. Thus, detection of social aspects and signals are crucial in order for a robot to fulfill the role of a human bartender. For example, the robot should serve the guests in the right order. Furthermore, in order to interact with humans in a more indirect but similarly important way, he should also be able to recognize the activities of human actors so that he can interact with them at any time depending on the situation. For example, if two persons are clinking glasses, the robot could respond with ”Cheers!”, for instance.

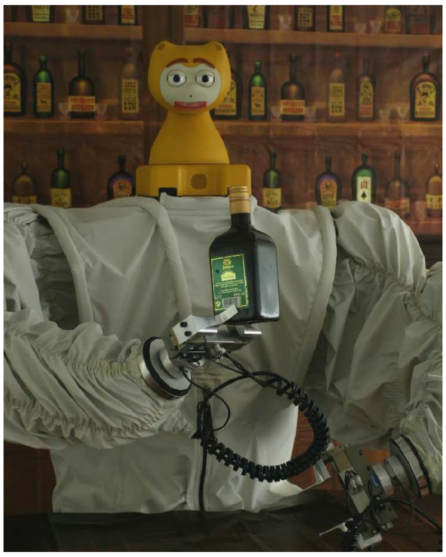


Figure 3.1: **JAMES robot** - robot acts as a bartender in a bar scenario serving humans.

3.2 Bar scenario description

In the context of this bar scenario, let us now think of appropriate activities. Throughout the thought process, we will entitle them in brackets.

A person can enter (ENTER) and leave (LEAVE) the area in front of the bar. To

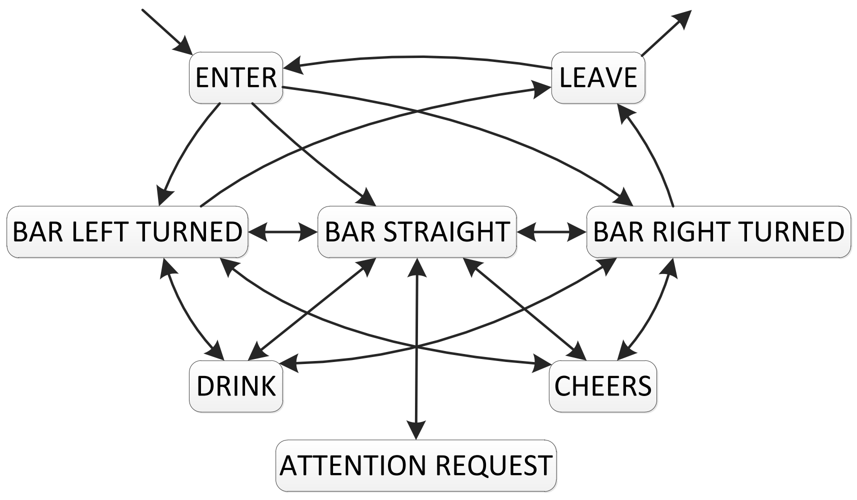


Figure 3.2: **Resulting state transition diagram of our bar scenario** - We have eight states and 26 transitions, including the start transition to Enter and the final transition out of Leave.

recognize where his or her main attention is directed to, we consider tracking the orientation of the body and hence if he is turned to the left (Bar Left Turned), to the right (Bar Right Turned) or straight to the bar (Bar Straight). Note that here we are not so much interested in the position of the body as it can easily be computed analytically, for instance by calculating the center of mass of all the joints. Near the bar, one can request the attention of the bartender (Attention Request). Evaluation of Kerstin Huth's video material of guest-bartender interactions as presented in [13] has shown that guests often physically request the attention by leaning forward towards the bartender, more often than using gestures, thus allowing us to track this state with our skeleton data. As a matter of course, a person will drink near the bar (Drink), either with the left or with the right hand. Furthermore, as some kind of social activity, one can clink glasses, drink a toast or just say cheers (Cheers) towards other guests or the bartender, with either the left or the right hand.

In the next step, we will define adequate transitions between these eight activities. From now on, we will denote them as our states.

At the beginning of any scenario, the person has to be calibrated by the Kinect and thus standing outside of the bar area. That said, we will always start with Enter, the so called start state or start node. We then assume that sooner or later our person will enter Bar Left Turned, Bar Straight or Bar Right Turned. In each of these three body orientations, Drink and Cheers are possible new states. Within these five states, we can transition among one another, except for the following two cases: Firstly, turning from left (Bar Left Turned) to right (Bar Right Turned) and vice versa obviously requires Bar Straight in between. Secondly, we forbid direct transitions between Drink and Cheers. Although we could label our data in a different way allowing direct transitions, practice has shown that our model becomes more stable disallowing them, as there is always skeleton data between these two states belonging to either Bar Left Turned, Bar Straight or Bar Right Turned, concerning the latter case. However, to continue, Attention Request, as we defined it by leaning forward towards the bartender, should only be considered when being in the state Bar Straight. Also, each end of Attention Request leads us again into Bar Straight. So far, we just have to discuss transitions involving the state Leave. In order to leave the bar area, we assume that our person first has to turn to either the left or the right side, because in practice no one leaves the bar moving backwards. Furthermore, we suppose that leaving the bar area does not occur immediately after Drink or Cheers, as there should always be at least one frame of skeleton data belonging to either Bar Left Turned or Bar Right Turned before leaving. That said, we only allow transitions from Bar Left Turned and Bar Right Turned to the state Leave. At last, being in Leave, we can re-enter (Enter) the bar area or complete our scenario, hence Leave will be our end state or end node. Finally, Figure 3.2 shows the resulting state transition diagram we will implement later using a hidden Markov model.

4 Implementation

After describing our bar scenario, we can now turn to the implementation part.

In this chapter, we mainly discuss the Kinect hardware and the software tool HTK and its integration in a software program called ”Kinactivity”. Thereby, we will focus on understanding the individual working processes when utilizing Kinactivity. With these tools, we can build, train and test our hidden Markov model. The results gathered in the process can then be evaluated in the next chapter.

4.1 Hardware: Kinect-Sensor description and technical aspects

Our motion sensing input device to provide us with joint data will be the Kinect by Microsoft. As described more detailed in [9, 22, 31], let us now gain a quick overview of the Kinect.

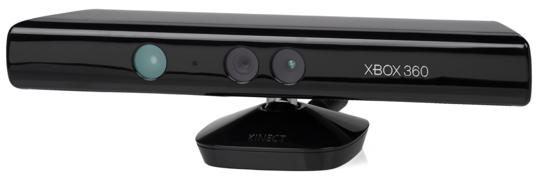


Figure 4.1: **Picture of the Microsoft Kinect** - Sensing input device equipped with the IR projector, Color- and Depth Camera (left to right)

Launched in November 2010, Microsoft originally released the Kinect for the Xbox 360 video game console providing a new way of interacting with the Xbox, where your body is the only input device.

The Kinect consists of a RGB-Camera (VGA, 640 × 480 pixels, 30 Hz), an IR Depth-Camera (QVGA, 320 × 240 pixels, 30 Hz), an IR projector and a multi-array microphone allowing acoustic source localization and ambient noise suppression. The depth sensing is based on the ”Light Coding”-technique by the israeli company PrimeSense and works as follows [22]: While the infrared laser (IR projector) projects a structured dot pattern (infrared laser grid) similar to a starry sky, the Depth-Camera, equipped with a monochrome CMOS-sensor, receives the infrared light reflected by the scene. The depth map is generated by comparing this resulting picture with the stored reference pattern. The depth sensor can maintain tracking through approximately 0.7 to 6 m, but the practical ranging limit is of 1.2m to 3.5m distance.

Build upon this structured-light approach to receive a 3D image, the Kinect is equipped with software allowing revolutionary full body tracking of multiple persons at low financial costs. Whereas range cameras (3D cameras) were very expensive (thousands of euros), thanks to large-scale production, the Kinect's price has reached around 100 €. According to UBM Techinsights, the total costs of the Kinect components are about 56 US-Dollar, as stated in [22]. Furthermore, the Kinect holds the Guinness World Record of being the "fastest selling consumer electronics device" [14].

As stated in [21], the Microsoft Research laboratory applied machine learning techniques and a very large database of pre-classified images, covering varied poses and body types, to develop a motion-capture solution in real time without the need of any required instrumentation (e.g. markers) placed on the moving human subject. This work was awarded by the 2011 MacRobert Award for engineering innovation.

In December 2010, PrimeSense released their own open source drivers allowing the Kinect to be effectively used in computer environments. PrimeSense has partnered with Willow Garage and Side-Kick to create OpenNI, a not-for-profit organization set up to "certify and promote the compatibility and interoperability of Natural Interaction (NI) devices, applications and middleware" [12]. The OpenNI-framework provides a uniform programming interface for driver- and middleware-modules allowing tasks such as (skeletal) tracking and visual analysis of the sensor data. The NITE-middleware by PrimeSense allows the real-time motion tracking of persons providing skeletons with 15 joints each.

A short time after the release of the Kinect, applications were programmed, which go far beyond the system's intended purpose of playing games: Human-Computer Interaction (HCI) using gestures and spoken commands (natural user interface), interaction between robots and humans, medical visualization, virtual-reality applications, input device for 3D printers, interactive media, and many more.

On June 2011, Microsoft has released an official non-commercial Kinect software development kit (SDK) for Windows, which paves the way for many new sophisticated high-level applications of this broad range of relatively new research areas.

In our bar scenario, however, we will use the position of nine joints obtained by the Kinect: the head, neck, torso and (each left and right) the shoulder, elbow and the hand. This results in 27 continuous position values, as we consider the x, y and z position of each joint, so that our HMM observation vector will have 27

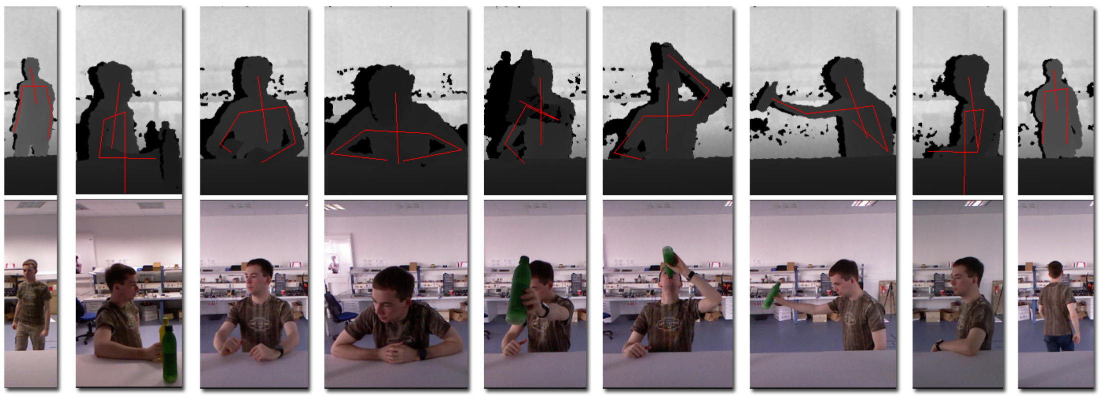


Figure 4.2: **Instances of skeleton poses belonging to the possible states in our bar scenario** - From left to right: ENTER, BAR RIGHT TURNED, BAR STRAIGHT, ATTENTION REQUEST, CHEERS (to the bartender with the left hand), DRINK, CHEERS (to the left with the right hand), BAR LEFT TURNED, LEAVE.

dimensions.

Figure 4.2 shows resulting skeletons and examples of skeleton poses belonging to the possible states in our bar scenario.

4.2 Software: HTK

We use the Hidden Markov Model Toolkit (HTK)[1] for building, learning and testing HMMs. Main source for utilizing HTK is the HTKBook [33]. HTK is primarily designed for creating HMM-based speech recognition tools and therefore much of the infrastructure support is dedicated to this task, as stated in the HTKBook [33]. However, with various adjustments and some overhead, we can use HTK as a powerful toolkit for our purposes with no need to implement the algorithms discussed in Chapter 2 on our own. Nevertheless, we have to be aware of how the underlying algorithms work in order to effectively deploy HTK tools in practice. Sotzek [26] laid a solid foundation for the use of HTK with non-speech related data.

Before actually discussing the HTK tools we will use for training, testing and analysis purposes, we first have to go through some data preparation steps in order for us to implement our bar scenario problem in HTK.

[1] `http://htk.eng.cam.ac.uk/`

4.2.1 Data preparation

In section 3.2, we introduced our bar scenario with eight states (activities) and 26 transitions. However, in HTK, we model and later train each activity as an own isolated HMM and combine them according to a so called defined HMM network. In the end, we obtain one final composed HMM.

For each of the activity models, we first have to create an initial prototype model using the HMM definition language described in Chapter 7 of the HTKBook [33]. Let us first think of an appropriate topology for these individual models. In section 2.2.1, we introduced left-to-right models to account for processes whose properties change over time. In case of activity recognition, and similar the workflow for assembly tasks as presented in Sotzek [26], left-to-right models can represent the actual development of the activity process over time, so these are appropriate models for our task. We implement them by setting the respective elements of the transition matrix **A** to zero initially, that way they will remain zero throughout the process of the learning algorithm. For each state, we allow a maximum of one state which can be jumped over when transitioning.

Furthermore, we have to think of the number of inner states of each activity model and of the mixture count, i.e. how many multivariate Gaussians are involved in generating the output in a specific state. These will be two crucial parameters and as there is no analytical way to determine the best numbers, we will later in the evaluation part come back to this problem. However, we make two assumptions in order for us to lower the free parameter: For each activity, the number of inner states, and for every inner state of each model, the mixture count is equal. Thus, we will have to determine two global parameters later on: The number of (inner) states and the number of Gaussians. From now on, mentioning the number of states will refer to the inner states, whereas the number of activities is always fixed to be eight.

Note that in HTK, the entry activity model is determined by the HMM network and the entry state of each activity itself is always the first inner state, as we use left-to-right models. Moreover, for each model, the first and the last state (which is the exit state) are non-emitting, because of the composition of all the models. Thus, having a state count of five, for instance, results in just three ”real” states which actually are able to generate output.

In order to combine these single activity models to a composed HMM, in HTK, we make use of the HMM network defined by the HTK Standard Lattice Format (SLF), described in the HTKBook [33], Chapter 20. In our bar scenario, we can directly use our previously modeled diagram 3.2 with 26 transitions to create a file called MODEL_NET.SLF manually.

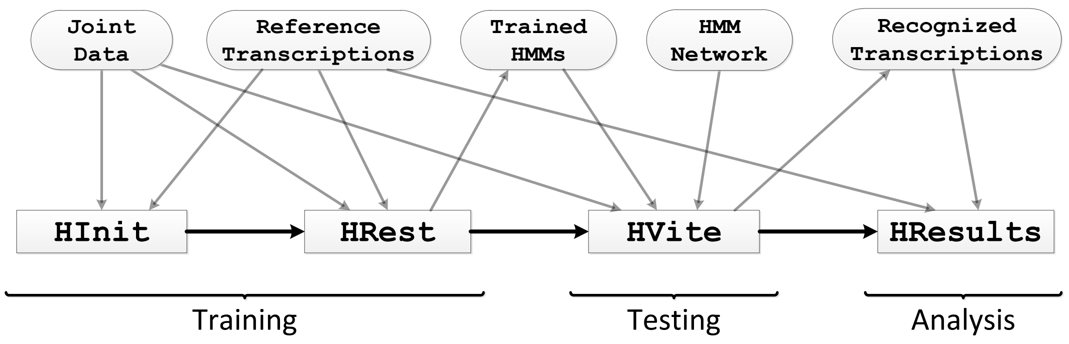


Figure 4.3: **Processing stages with our four HTK tools** - In the training process, we pass the joint data and the reference transcriptions to each HInit and after it HRest to generate the trained set of HMMs. With them, our joint data and the HMM network, i.e. the structure of composed activity-HMMs, we can invoke HVite to build the final recognized transcription files. This process is called the testing phase. In the end, we can compare the reference with the recognized transcriptions invoking HResults in order to analyze the performance of our trained HMM. This figure is inspired by HTKBook [33, p. 17].

Finally, note that label files (transcription files) are stored using the HTK Master Label File (MLF) format, as described in the HTKBook [33], Chapter 6.

4.2.2 HTK Tools

Out of the over 30 command line HTK tools, we just need the following four to solve our HMM based activity recognition task: HInit, HRest, HVite and HResults. They will later be applied in just the same order we have listed them. You can gather more information about HTK tools in chaper 17 of the HTKBook [33], particularly for these tools in section 17.9, 17.18, 17.23 and 17.19, respectively. Now let us briefly discuss what these tools actually do.

4.2.2.1 HInit

A quick recap: In order to apply the learning algorithm for HMMs, i.e. the EM algorithm discussed in Chapter 2, we need an initial selection for the model parameters λ^{old}. HInit does exactly this estimation process. That said, at the beginning of the training stage, we first invoke HInit for every activity model as the following instance shows:

```
HInit -A -D -i 75 -S model_train_data.txt -M hmm/ -H proto/hmm_drink
-I model_train_labels_true.mlf -l drink drink.
```

The basic principle of HInit is realized similar to the iterative learning algorithm, but instead of assigning each observation vector to every state according to the probability of being in that state (i.e. considering $\gamma(X_t = S_i)$), we assign each observation vector to one individual state, starting with dividing them equally among all states (uniform segmentation). HInit then makes use of a specific segmental k-means procedure computing means and variances of each mixture by averaging the observation vector of each state, applying the Viterbi algorithm to find the most likely state sequence to finally reassign the observation vectors (Viterbi alignment). We then start over again until the estimated values do not change (convergence) or some maximum number of estimation cycles has been reached, in our example above we set this number to 75. Note that with each iteration, the transition probabilities are estimated by considering the number of times the Viterbi alignment makes each transition. HInit is explained in detail in the HTKBook [33], sections 1.4, 8.2 and 17.9.

4.2.2.2 HRest

In practice, however, we have to discard the assumption that each observation vector can be assigned to one specific state. Thus, HRest further re-estimates the initial model parameter values computed by HInit by replacing the segmental k-means procedure with the EM learning algorithm we discussed in Chapter 2. Again for every activity model, we invoke HRest with the following command:

```
HRest -A -D -i 75 -S model_train_data.txt -M hmm/ -H hmm/macros -H
hmm/hmm_drink -I model_train_labels_true.mlf -l drink drink.
```

HRest is explained in detail in the HTKBook [33], sections 1.4, 8.4 and 17.18.

4.2.2.3 HVite

Now that we obtained our final trained HMM for each activity, with our HMM network, we can decode the most likely series of activities for a given sequence of joint data based on the Viterbi algorithm discussed in Chapter 2. This task is performed by the recogniser or decoding tool HVite, which outputs the resulting recognized transcriptions of the input joint data. We invoke HVite once as follows:

```
HVite -A -D -T 4 -S model_train_data.txt -H hmm/macros -H
hmm/hmmdefs -i model_train_labels_recognized.mlf -w model_net.slf
```

```
model_dict.txt model_list.txt.
```

This allows us to actually test the performance of our final model concerning the testing and training data. Again, HVite is explained in detail in the HTKBook [33], sections 2.3.3, 17.23 and especially in Chapter 13.

4.2.2.4 HResults

In order to analyze the performance of our recogniser (trained HMMs), we need to compare the recognized with the corresponding reference transcriptions. In HTK, we make use of HResults, a performance analysis tool which provides various meaningful statistics. HResults can be invoked as follows:

```
HResults -f -p -I model_train_labels_true.mlf model_list.txt
model_train_labels_recognized.mlf.
```

Among various statistics, based on the standard US NIST FOM metric, HResults counts substitution, deletion and insertion errors and hence can compute the activity accuracy. We will come back to the possibilities of this tool and discuss the statistics in Chapter 5 when we evaluate our bar scenario results. More information about this tool can be found in the HTKBook [33], sections 2.3.4, 3.4, 13.4 and 17.19.

4.2.3 Final notes

Figure 4.3 illustrates the processing stages of the four HTK tools we have just discussed.

Finally, let us briefly mention two more HTK tools, which will not be necessary in order to create, train and test our model, but are still embedded in our software program Kinactivity.

At first, HSGen is a useful tool which generates random examples of our defined HMM network in order to somewhat verify the correctness of the network or to obtain sample sequences of activities which can be used as a basic structure for training scenarios. Sections 12.6 and 17.20 in the HTKBook [33] provide more information about HSGen.

Instead of creating the HMM network manually as we mentioned above, HTK also provides a way to build the network automatically given a grammar based on extended Backus-Naur Form (EBNF) notation. The tool is called HParse and further information about it and about EBNF can be found in the HTKBook

[33], sections 12.3 and 17.16. However, in our scenario, we use a manually created HMM network.

Note that, in our software program Kinactivity, every time we execute a HTK tool, we will store the output of that tool in a corresponding log file. Thus, if any errors occur, we can easily trace them.

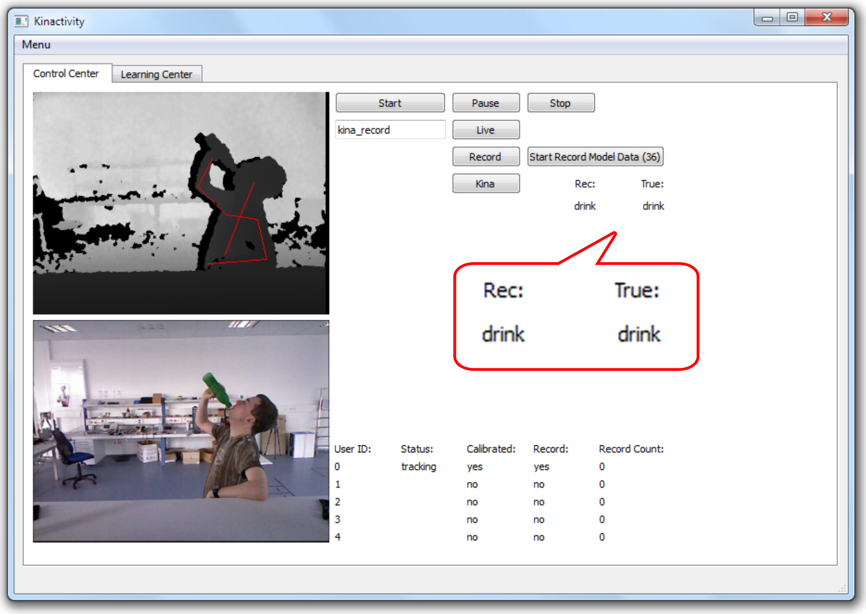
4.3 Software: Kinactivity

In order to record data, create, train and test HMMs, it is much more comfortable to have a single software program that automatically handles most of the processes and tasks needed. In the end, the user should only be confronted with the essential actions and decisions. That is why we developed Kinactivity, a software programmed in C++, which makes use of the following libraries and frameworks:

- OpenNI[2], a framework for writing applications utilizing natural interaction. In Kinactivity, this API covers communication with the Kinect, as well as the high-level user tracking with NITE. We make use of the OpenNI Binaries (i.e. ”OpenNI Unstable Build for Windows x86 (32-bit) v1.3.2.3 Development Edition”), the OpenNI Compliant Middleware Binaries (i.e. ”PrimeSense NITE Unstable Build for Windows x86 (32-bit) v1.4.1.2 Development Edition”) and extern[3] OpenNI Compliant Hardware Binaries (i.e. ”SensorKinect-Win-OpenSource32-5.0.3.4.msi”).
- Qt[4] 4.7.4, a framework for developing application software with a graphical user interface (GUI). Here, Qt provides a helpful and interactive GUI for Kinactivity.
- OpenCV[5] 2.3.1, a library of programming functions for real time computer vision. It is used primarily for manipulating (e.g. resizing) the depth and image data and then transferring them from the OpenNI framework to Qt.
- Boost[6] 1.47.0, free peer-reviewed portable C++ source libraries. Kinactivity utilizes them in many cases, especially concerning filesystem, multi-thread synchronizing and string algorithm issues.

Every library and framework can be used platform-independent. However, due to the software development process, we have just tested Kinactivity on Windows

[2]`http://www.openni.org/`
[3]`https://github.com/avin2/SensorKinect`
[4]`https://qt.nokia.com/`
[5]`http://opencv.willowgarage.com/wiki/`
[6]`http://www.boost.org/`



User ID:	Status:	Calibrated:	Record:	Record Count:
0	tracking	yes	yes	0
1		no	no	0
2		no	no	0
3		no	no	0
4		no	no	0

Figure 4.4: **Screenshot of Kinactivity's Control Center** - The user can see the RGB (left lower one) and the depth image (left upper one), record data from our Kinect and label them. Thereby, basic start, stop and pause buttons are implemented and when viewing labeled files (special data format *.kina), one can compare the recognized (here denoted as "Rec:") and the reference ("True:") transcriptions over time.

so far. Nevertheless, porting it to Linux does not imply a great effort and will be eventually done in the future.

Kinactivity is divided into two tabs: the "Control Center" and the "Learning Center". Let us now take a deeper look at both of them.

4.3.1 The Control Center

In the "Control Center" (Figure 4.4), on the upper left side you can see the depth image and below the RGB image. If users are tracked, the depth image also displays the skeleton for each user in another colour. On the upper right side you can find the control buttons. "Start" and "Pause" should be self-explanatory. Below, you can switch to another source. There are three possible sources: "Live", "Record" and "Kina".

To make the live mode work, you need a functioning Kinect connected with your PC and OpenNI drivers installed. During the live mode, RGB and depth data are recorded using the OpenNI recording file format (*.oni). Every recording starts with selecting "Live", is stored in Kinactivity's workpath in the directory "data/"

using the file name specified by the input field next to the button, and ends with selecting another source or by shutting down the program.

In the record mode, you can replay any *.oni file. Only in this mode it is possible to collect training data or test data used for our hidden Markov model by clicking on ”Start Record Model Data”. The number in brackets indicates which data set you are actually recording and increases each time you start and stop recording model data. Keep in mind that model data is only recorded when at least one user is calibrated and being tracked. The recorded model data consists of several files which again are initially stored in Kinactivity’s workpath directory ”data/”. Recording will stop when switching to another source, selecting ”Stop Record Model Data” after starting, when the record ends or by shutting down the program. After recording, you can choose whether you want to train or test the HMM with that data, and, depending on the choice, move the files to either ”data/train” or ”data/test”.

The ”Kina”-mode is by far the most interesting mode for visualizing model data and finally the output of our hidden Markov model. For each recording, a metafile (*.kina) is generated. With the ”Kina”-button, we can select any kina-file in either ”data/train” or ”data/test” and replay the record showing the current reference and (if transcription files were generated in the Learning Center) the current recognized activity.

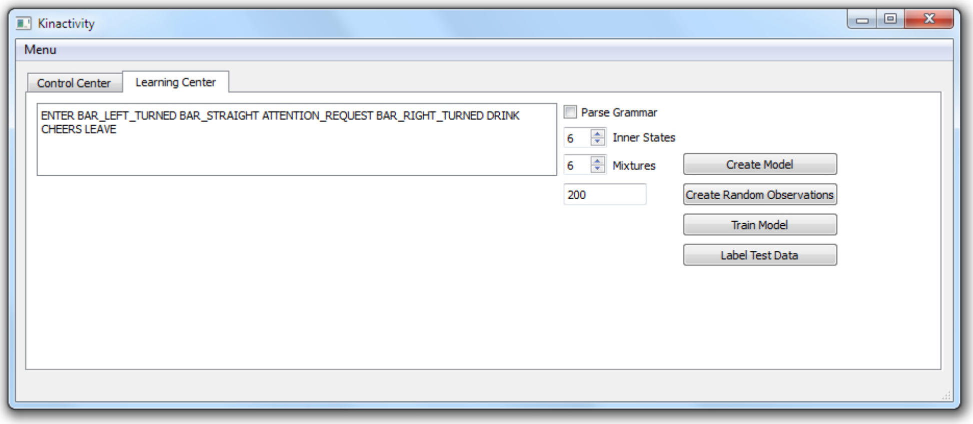
Figure 4.4 shows a screenshot of the Control Center playing a kina-file. The red caption illustrates the current reference and the current recognized activity.

4.3.2 The Learning Center

The Learning Center allows us to create, train and test HTK related HMMs of any scenario. Figure 4.5 shows the layout of this tab.

At first, we have to develop an initial model. Depending on whether we are creating the HMM network manually or use the EBNF notation to generate it automatically, we just need to insert the names of the activities (in capital letters, separated by a space) or the complete grammar containing the activity names into the text field on the upper left side. Furthermore, before pressing the ”Create Model”-button, we need to determine the inner state and the mixture count concerning each HMM of the eight activities. Then, when pressing the button, the following files will be created:

- `model_gram.txt` stores the string in the text input box and is used by several HTK tools



ENTER BAR_LEFT_TURNED BAR_STRAIGHT ATTENTION_REQUEST BAR_RIGHT_TURNED DRINK CHEERS LEAVE

☐ Parse Grammar
6 Inner States
6 Mixtures
200

Figure 4.5: **Screenshot of Kinactivity's Learning Center** - Here the user can actually create a HTK related HMM based upon the states in the text field (or the grammar if parse is checked) and the number of inner states and mixtures. One can also create as many random observations as one wishes in order to test the HMM network structure, for instance. Furthermore, we can train the HMM and subsequently label all training records based on the trained HMM, and write the transcriptions concerning the test records.

- `model_list.txt` contains the list of HTK-related phonemes, which are the names of activities to lowercase letters
- `model_dict.txt` contains the dictionary, i.e. a list of activities and their orresponding phoneme
- `model_gram.txt` contains the HMM network [only if parse grammar is checked]
- HMM prototype files for each activity in the subfolder "model/proto/"

Invoking "Create Random Observations" creates the file `model_random_cases.txt` which contains as many random examples of possible activity sequences as is specified in the input box on the left.

After initializing our model, we can train it by pressing the "Train Model"-button. The underlying course of action is as follows:

1. Create `model_train_data.txt` to list all joint data files (.bin) in "data/-train"
2. Create `model_train_labels_true.mlf` containing all transcription files (.lab) in "data/train"
3. Invoke HInit for each activity HMM to estimate initial model parameters
4. Create the global options file `macros` in "model/hmm/" containing several

meta data information

5. Invoke HRest for each activity HMM to apply the EM learning algorithm
6. Create the Master Macro File (MMF) `hmmdefs` storing all the HMM definitions in one file
7. Invoke HVite to generate `model_train_labels_recognized.mlf`, the Master Label File (MLF) containing all the recognized transcriptions of the training data
8. Invoke HResults to gather statistics about the performance of our final model on the training data, in which the output is stored in the file `HResults_train.log` in ”model/log”

Furthermore, the ”Label Test Data”-button invokes the following:

1. Create `model_test_data.txt` to list all joint data files (.bin) in ”data/test”
2. Create `model_test_labels_true.mlf` containing all transcription files (.lab) in ”data/test”
3. Invoke HVite to generate `model_test_labels_recognized.mlf`, the Master Label File (MLF) containing all the recognized transcriptions of the test data
4. Invoke HResults to gather statistics about the performance of our final model on the test data, in which the output is stored in the file `HResults_test.log` in ”model/log”

5 Evaluation

After discussing how to implement our HMM based approach to recognize activities, we can now turn to the evaluation of our results.

For our bar scenario, we have recorded 34 real scenarios and about 11 MB of raw joint data resulting in about 106,000 data frames and hence about nearly 60 minutes of recording. Altogether, we manually labeled 2307 activities for our reference transcription files. In order to evaluate the results, we divide them into 70% train and 30% testing data, so that 20 scenarios (1612 activities) are used for training and 14 (695 activities) for testing purposes. To measure the performance of our final HMM, we are most interested in maximizing the recognition of the test data, as they can tell us, if our model generalizes successfully and can actually recognize activities it has not been trained with.

For the evaluation of our results, we make use of the HTK performance analysis tool called HResults, further described in the HTKBook [33], Chapter 17, HResults (17.19). HResults compares the recognized transcription files created by HVite with the referenced ones we labeled manually. Thereby, it uses the standard US NIST FOM metric to generate recognition statistics. We will now discuss this process in detail.

Let us first take a look at an example of a recognition statistic obtained by HResults for a 6-4-HMM (6 inner states, 4 mixtures):

```
---------------- Overall Results -----------------
SENT: %Correct=0.00 [H=0, S=14, N=14]
WORD: %Corr=88.78, Acc=68.63 [H=617, D=42, S=36, I=140, N=695]
===================================================================
```

The first line denotes the percentage number of recognized label files (scenarios) which are completely identical to the referenced ones. Obviously, as our scenarios are quite long and difficult to recognize completely, there are no identical ones. Instead of comparing complete scenarios, we should better discuss comparing the actual activities (in HTK here called words). The second line gives us detailed information about this. First of all, let us take a look at the data in the brackets: Based on 695 labeled activities in total (denotes as N), 617 labels were correctly recognized (H), whereas 42 labels were deleted (D), 36 substituted (S) and 140 new inserted (I) in the recognized label files overall. With these numbers, we can then calculate the percentage number of labels correctly recognized (%Corr) and the accuracy (%Acc), as done by HResults using the following formulas:

$$\%\text{Corr} = \frac{H}{N} \times 100\% \tag{5.1}$$

$$\%\text{Acc} = \frac{H - I}{N} \times 100\% = 100\% - \left(\frac{D + S + I}{N} \times 100\% \right). \tag{5.2}$$

Note that every activity (label) in the reference transcription files can either be recognized correctly, substituted by another activity or just be deleted (skipped) in the recognized transcription files, so that $N = H + D + S$. In order to account for activities which were inserted in our recognition files in between substitutions, deletions or correct labels, we just defined (5.2) for a measurement of our accuracy. Thus, the accuracy is a more representative measure for the recogniser performance than the correctness value. The counterpart to the accuracy is the ratio of errors related to the total number of activities, as shown by the right term of the second equation in (5.2).

After briefly discussing the terminology involved in our evaluation, we can now turn to the question, how many inner states and how many Gaussians (mixture count) our model should have in order to gain the best results. As there is no formula to determine the best numbers, we have to apply the trial and error approach.

5.1 Variable mixture count

Let us begin with the mixture count. We will evaluate our testing and training datasets with a constant inner state count of six and a variable mixture count from 1 up to 10. In a short pre-evaluation process we made beforehand, an inner state count of six has shown good results, so in this section we will stick with it.

Figure 5.1 shows the resulting overall performance in a clustered column chart when evaluating (5.1) and (5.2) for both the testing (blue columns) and training (green columns) datasets.

First of all, we should reflect that we are most interested in maximizing the performance on our test data. Despite the fact that the performance on the training data increases with the mixture count, the accuracy of the testing data significantly decreases. Note that even with two Gaussians, the correctness and the

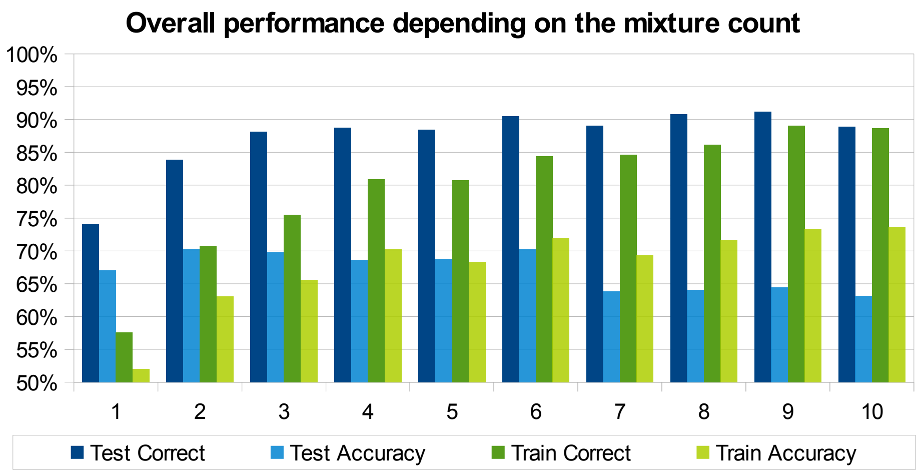


Figure 5.1: **Clustered column chart showing the overall performance with six states and a variable mixture count from 1 up to 10** - For each mixture count and both the testing and training datasets, we have illustrated the percentage number of labels correctly recognized (denoted as Test/Train Correct) and the accuracy (denoted as Test/Train Accuracy).

accuracy of the training data shows very good results. Let us now take a look at chart 5.2 to further analyze the impact of different mixture counts.

Figure 5.2 illustrates the composition of the recognized testing and training labels, i.e. how many of them were correct, how many reference labels were ignored (deletions) or substituted (substitutions) by another label and how many labels were added (insertions) without being in the reference label file. Here we can see that increasing the mixture count means decreasing the amount of deletions and substitutions, but increasing the insertions. In our bar scenario, for example, inserting Bar Right Turned, Bar Straight or Bar Left Turned when being in one of the two other states is sometimes absolutely reasonable, as there is no clear line between them. Thus, we value the accuracy of our models with more Gaussians as being slightly better than for example the one with two Gaussians.

However, to somewhat balance between a low mixture count to prevent overfitting, satisfactory performance on the testing data and also slightly considering the training data, we decide to pick a mixture count of 6.

5.2 Variable state count

Let us now briefly discuss the same procedure as in the last section, but now with a variable state count from 3 up to 10. As a result of the previous evaluation

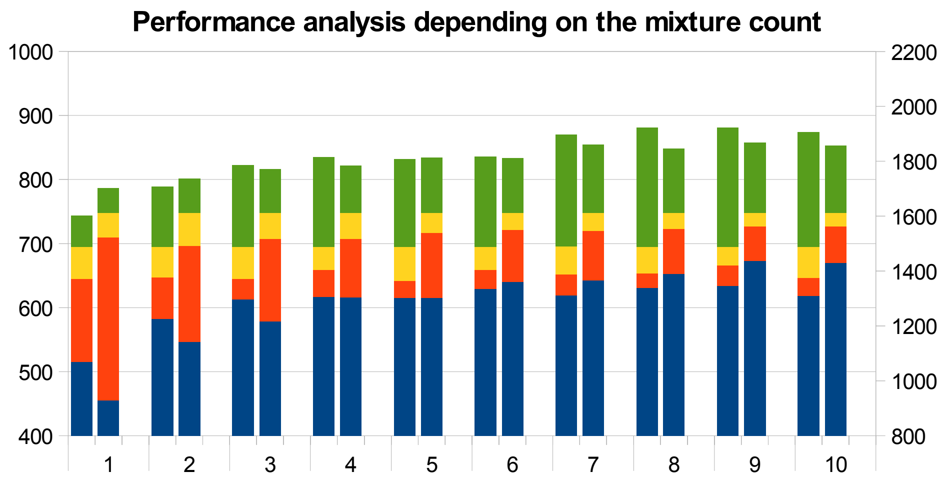


Figure 5.2: **Clustered-stacked column chart showing a more detailed performance analysis having a variable mixture count and six states** - For each mixture count and both the testing (left columns) and training (right columns) datasets, we have illustrated the exact number of correct labels (blue), deletions (orange), substitutions (yellow) and insertions (green). Note that the left y-axis shows the values concerning the testing results, whereas the right y-axis shows the values concerning the training results.

process, we will fix the mixture count to 6 and again invoke HResults with the testing and training data.

As in the previous section, the clustered column chart 5.3 shows the overall performance, whereas the clustered-stacked column chart 5.4 illustrates the composition of the recognized testing and training labels.

First of all, we can conclude that the accuracy when dealing with less than five inner states is significantly bad. Models like these correctly recognize many acitivities, but achieve this most likely by guessing, as the number of insertions indicates, compared to the other models. Generally, increasing the state count leads to less insertions, but slightly more deletions.

For us, a state count of 6 seems to be a good balance between a low state count to prevent over-fitting and a good performance on the testing data. Note that, with more testing data in the future, we have to recap the decision process for picking appropriate mixture and state counts.

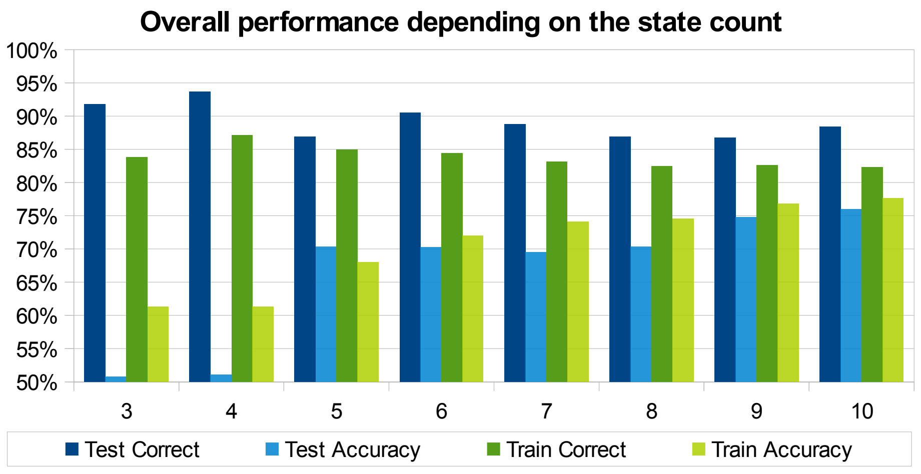


Figure 5.3: **Clustered column chart showing the overall performance with six Gaussians and a variable state count from 3 up to 10** - For each state count and both the testing and training datasets, we have illustrated the percentage number of labels correctly recognized (denoted as Test/Train Correct) and the accuracy (denoted as Test/Train Accuracy).

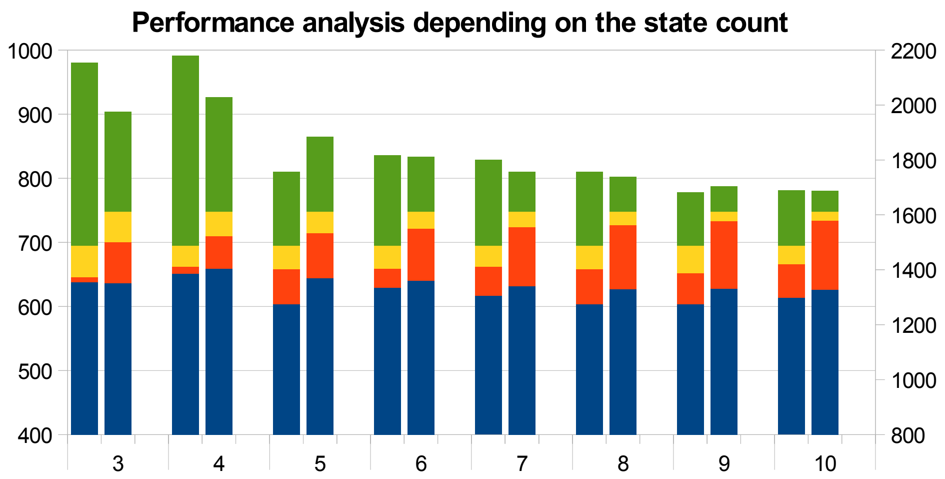
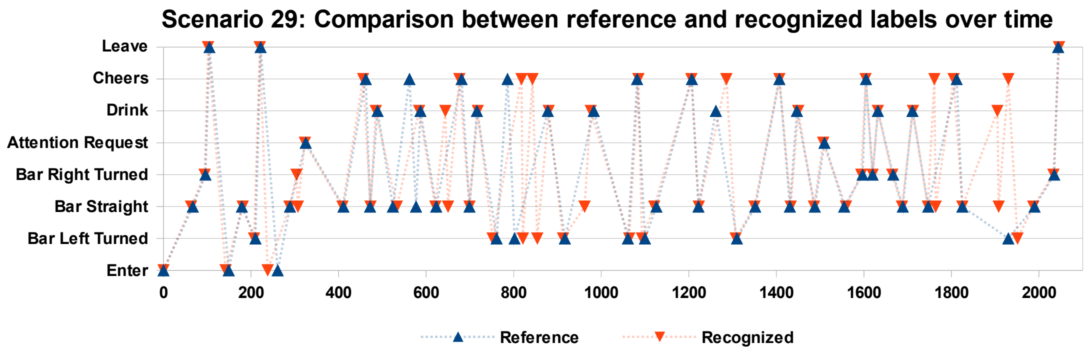


Figure 5.4: **Clustered-stacked column chart showing a more detailed performance analysis having a variable state count and six Gaussians** - For each state count and both the testing (left columns) and training (right columns) datasets, we have illustrated the exact number of correct labels (blue), deletions (orange), substitutions (yellow) and insertions (green). Note that the left y-axis shows the values concerning the testing results, whereas the right y-axis shows the values concerning the training results.



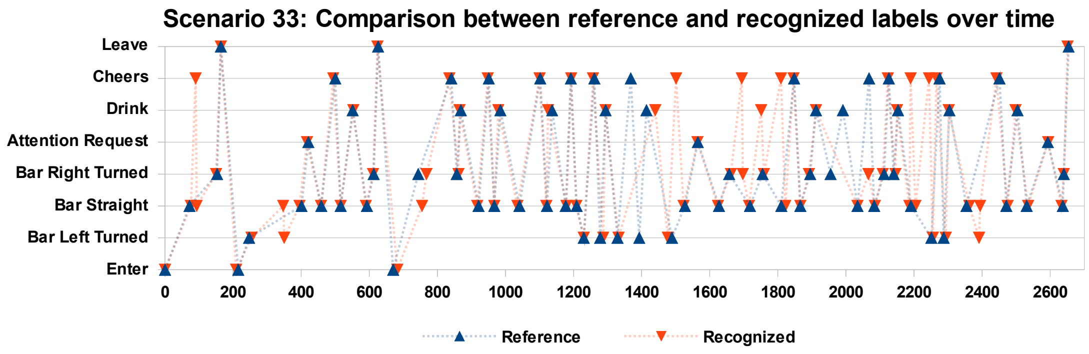


Figure 5.5: **Performance of our 6-6-HMM based on scenario #29 (upper diagram) and scenario #33 (lower diagram)** - We plot the true/reference labels (blue triangles) and the recognized labels (red triangles) over the course of time, more precisely over the data frames of each scenario.

5.3 Performance of the bar scenario model

In the previous sections we have argued that, within the pool of possible HMM structures, a HMM with 6 inner states and 6 Gaussians should give us the best results for our bar scenario. Let us now take a deeper look at the performance of our HMM by analyzing two scenario instances and classification statistics for each of our activities based on a so called confusion matrix.

Figure 5.5 shows a graphical evaluation of two of our scenarios intended for testing purposes by comparing the manually labeled (reference) activities with the recognized one over time. Thereby, scenario #29 has a correctness value of 96.55% with an accuracy of 79.31%, scenario #33 features 93.24% and 72.97%, respectively.

For the first time, we can reveal the exact timings between the reference and the recognized labels. It seems like they are most often quite consistent, especially when the person drinks or requests attention. They seem to differ more when the person turns, but this is acceptable, as there is a smooth transition between these

%Corr=90.50, Acc=70.22 [H=629, D=30, S=36, I=141, N=695]											
	atten	bar_l	bar_r	bar_s	cheer	drink	enter	leave	D	%c	%e
atten	47	0	0	0	0	0	0	0	0	100.0	0.0
bar_l	0	87	2	2	0	0	0	0	6	95.6	0.6
bar_r	2	4	52	4	1	1	0	0	7	81.3	1.7
bar_s	0	0	1	234	1	0	0	0	6	99.2	0.3
cheer	1	1	2	1	73	6	0	0	7	86.9	1.6
drink	0	1	2	1	3	96	0	0	4	93.2	1.0
enter	0	0	0	0	0	0	20	0	0	100.0	0.0
leave	0	0	0	0	0	0	0	20	0	100.0	0.0
I	3	21	13	51	26	25	1	1			

Figure 5.6: **Confusion matrix comparing the classification of our testing data between our reference labels (rows) and the recognized ones (columns) gained by our 6-6-HMM (headline)** - Whereas %c shows the percentage number of that activity being correctly recognized, %e indicates the percentage number of incorrect labels divided by all labels $N = 695$ and thus gives us a measurement of that activity being often recognized incorrectly. Furthermore, D and I illustrate the deletions of reference labels and the insertions of recognized labels, respectively, for each activity.

states. Furthermore, we can note that a reliable detection of the activity CHEERS seems to be the most difficult task for our HMM. However, recognizing the states ENTER, LEAVE and ATTENTION REQUEST is absolutely accurate.

Before analyzing and discussing the performance of our HMM too much, let us take a look at the confusion matrix[1] for our testing data presented in figure 5.6. The confusion matrix is generated by invoking HRESULTS with special flags, and further described in the HTKBook [33], Chapter 17, HResults (17.19). It visualizes the classifications done by our HMM by comparing them with the actual reference activities and hence easily shows how well we classify (or maybe confuse) activities and where our problems are.

At first, let us further discuss the structure and the data within the confusion matrix. The rows illustrate the reference activities we manually labeled. Thereby, the third-last column indicates for each row the number of activities which were not recognized by our HMM (deletions, denoted as "D"). Summing up all the rows, including the deletions, gives us the total number of reference labels, i.e. $N = 695$. In contrast, the columns show the number and classifications for the recognized activities. Thereby, denoted as "I", the last row includes the number of activities

[1] http://en.wikipedia.org/wiki/Confusion_matrix

that do not appear in the reference labels (insertions). An example to read the confusion matrix: Whereas three times our HMM recognizes Cheers, but the person was in Drink, six times the person clinks glasses but was recognized as a drinking person. Think in terms of substitutions when reading the table. Finally, the second-last column expresses (in percentage) how many times an activity was correctly labeled when dividing the number of correct labels in that row by the total number of instances in that row (not considering the deletions). Moreover, the last column indicates (again in percentage) the number of incorrect activities in that row divided by the number of all reference labels $N = 695$.

Now, let us see what information we can extract from this confusion matrix:

- Enter, Leave and Attention Request are recognized flawlessly, except for three insertions and very few substitutions concerning Attention Request and each one insertion for Enter and Leave. However, considering the occurrence of insertion errors, they can be explained by the fact that transitions are smooth and thus might cause jumps when being in between of two activities.

- Whereas Bar Straight and Bar Left Turned are recognized quite good, surprisingly, Bar Right Turned has the worst results of all activities. Most likely, this activity was most influenced by skeleton errors, wrong labeling or insufficient training data, as the execution and the recognition should be equivalent to Bar Left Turned. Moreover, the number of times we turned left in contrast to the one we turned right is not quite balanced in the test data (97 vs. 71).

- Besides Bar Right Turned, Cheers is the second worse recognized activity, as it is also the most complicated activity. Whereas 7 times it was not recognized at all, 6 times the person was assumed to drink. This can be explained with the fact that raise and stretching the arm near the head might be similar to some movements when drinking. Moreover, among the 104 times Cheers was recognized, the chance is 25% that it was inserted wrongly, which is the highest insertion error rate of all the activities.

- In contrast to Cheers, Drink is recognized much better. However, again, the insertion errors seem to be the part which should worry us the most.

- All in all, the number of insertion errors per activity seem to be more or less proportional to the number of times that this activity occurs in the test data.

5.4 Conclusion and future work

This bachelor's thesis presents a hidden Markov model (HMM) based approach to recognize activities of human actors interacting with a robot. Within the JAMES project, we consider a bar scenario in which the robot acts as a bartender and serves humans. With the joint data of these persons obtained by the Kinect, we can use the Hidden Markov Model Toolkit (HTK) and a software program Kinactivity to model, train and test a HMM to fulfill this task.

After the training process using 70% of our data, we can evaluate the results based on the remaining 30% of test data. Overall, with six inner states and six Gaussians contributing to the observation vector of each state, we recognize about 90% of the activities, whereas the accuracy, which also consideres activities inserted by mistake (insertion errors), is over 70%. This performance is in the same order of magnitude as the training results (84% and 72%). Thus, we successfully trained a fairly reliable and robust HMM which is capable of generalization and recognizes activities it has not been trained with.

However, there are two main problems affecting our results apart from any HMM structure or design problem. First of all, of course, the Kinect does not provide perfect joint data. Whenever there are obstacles between the Kinect and the person being tracked, e.g. bottles or glasses on top of the bar, or when we rotate 90 degrees and thus standing sideways, or even when the clothes of the tracked person are inappropriate, the skeleton becomes instable, even in many other situations. This leads us to the second problem. When we label the joint data and face situations where the Kinect provides an instable skeleton, we often can not label these frames correctly. Moreover, only having the 2D interface, we have no idea what the Z position of the joints are. However, even when the skeleton is stable, labeling the data manually can not be perfect. Often, if there are smooth transitions between our activities, as it is in our bar scenario concerning for example Bar Straight, Bar Left Turned and Bar Right Turned, we can not ensure to label them in a consistent way over time.

Another problem is that we need a sufficient amount of data. In this thesis, we mainly arranged the bar scenarios with one and the same person. However, even with a fixed skeleton proportion (i.e. one human actor), for example Cheers has many ways to be executed: We can clink glasses at any position in front of the bar, with any body alignment, stretch the arm to any direction, with either the left or the right hand. We guess that a HMM will need more data than we have gathered in order to also account for the differences between the behavior and the skeleton of different persons. Note that we also need a balance in our training data, i.e. we should have roughly the same amount of Bar Left Turned and

Bar Right Turned activities.

Despite these problems, the evaluation of our bar scenario results has shown that many even difficult activities and transitions can be recognized in principle, for example, if the person turns around while drinking and has another body alignment in the end. Furthermore, drinking and cheering with either the left or the right hand was quite often successfully recognized.

For future work, we also suggest to further adapt the structure of our activity models. At the beginning, we made the assumption that for every activity model, we use the same amount of inner states and the same amount of Gaussians. In practice, however, every activity is different and should be considered independently. Of course, this leads to many new parameters, but we might consider dividing activities in different classes. Furthermore, HTK provides many more adjustments which should be explored. In the end, we should also implement a live mode in order to recognize activities immediately. However, HTK does not provide an easy way for us to implement this, so we have to turn to other frameworks, most likely on top of HTK.

To conclude, this thesis provides initial research in activity recognition within the JAMES project, based on a bar scenario. We hope to lay a foundation for further investigation and improvement in this area of research and hence to help robots to interact with humans in a more social way. For example, when the robot detects that a person is drinking, clinking glasses or is saying cheers in front of the robot, he could respond by saying ”Cheers!”, so to stimulate the interaction and helping to strengthen the bond between the robot and the human. Furthermore, establishing new ways of human-robot interactions might provide new research areas concerning the acceptance of robots in an environment dominated by humans so far.

Bibliography

[1] Htk. http://htk.eng.cam.ac.uk/, September 2011.

[2] Jahir - joint-action for humans and industrial robots. http://www6.in.tum.de/Main/ResearchJahir, October 2011.

[3] Leonard E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[4] H. Berndt and K. Dietmayer. Driver intention inference with vehicle onboard sensors. *Vehicular Electronics and Safety (ICVES)*, 2009 IEEE International Conference:102–107, 2009.

[5] H. Berndt, J. Emmert, and K. Dietmayer. Continuous driver intention recognition with hidden markov models. *Intelligent Transportation Systems, 2008.*, ITSC 2008. 11th International IEEE Conference:1189–1194, 2008.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer Science + Business Media, LLC, 8 edition, 2006. http://research.microsoft.com/~cmbishop/PRML.

[7] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61:268–278, March 1973.

[8] fortiss GmbH. James - joint action for multimodal embodied social systems. http://www.fortiss.org/en/research/projects/james.html, October 2011.

[9] Hartmut Gieselmann. *Tanz der Skelette: Bewegungserkennung mit Kinect.* c´t, March 28, 2011. www.ct.de/1108100.

[10] Md. Kamrul Hasan', Husne Ara Rubaiyeat2, Yong-Koo Lee', and Sungyoung Lee. A reconfigurable hmm for activity recognition. *Advanced Communication Technology, 2008. ICACT 2008.*, 1:843–846, 2008.

[11] Homepage. *HTK.* http://htk.eng.cam.ac.uk/.

[12] Homepage. *OpenNI organization.* September 16, 2011. http://www.openni.org/.

[13] Kerstin Huth. *Wie man ein Bier bestellt.* Master's thesis, Universität Bielefeld, September 2011.

[14] Tim Ingham. *Kinect cruises past 10m sales barrier.* CVG, March 9, 2011. `http://www.computerandvideogames.com/292825/kinect-cruises-past-10m-sales-barrier/`.

[15] Xiaofei Ji and Honghai Liu. View-invariant human action recognition using exemplar-based hidden markov models. *Lecture Notes in Computer Science*, 5928/2009:78–89, 2009.

[16] Claus Lenz, Suraj Nair, Markus Rickert, Alois Knoll, Wolgang Rösel, Jürgen Gast, Alexander Bannat, and Frank Wallhoff. *Joint-Action for Humans and Industrial Robots for Assembly Tasks. Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication*, 1-3:130–135, 2008.

[17] Claus Lenz, Alice Sotzek, Thorsten Röder, Helmuth Radrich, Alois Knoll, Markus Huber, and Stefan Glasauer. *3D Occupancy Grids and HMMs for Human-Robot Workflow Analysis.* IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011), March 2011.

[18] George Soules Leonard E. Baum, Ted Petrie and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, February 1970.

[19] J. A. Eagon Leonard E. Baum. An inequality with application to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73:360–363, 1967.

[20] Ted Petrie Leonard E. Baum. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6): 1554–1563, 1966.

[21] PE. *Kinect team scoop the £50,000 MacRobert Award.* June 7, 2011. `http://profeng.com/news/kinect-team-scoop-the-50000-macrobert-award`.

[22] Patrick Pogscheba and Christian Geiger. *Erwachsene ebenso: Alternative Nutzung von Microsofts Kinect-Sensor.* iX, March, 2011. `www.ix.de/ix1103114`.

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

[24] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 3 edition, 2010.

[25] Dairazalia Sanchez, Monica Tentori, and Jesus Favela. Hidden markov models for activity recognition in ambient intelligence environments. *Eighth Mexican International Conference on Current Trends in Computer Science (ENC 2007)*, enc:pp.33–40, 2007.

[26] Alice Sotzek. Arbeitsflusserkennung für komplexe montagearbeiten mittels hidden markov models: Interpretation von multimodalen biologischen signalen. Master's thesis, TU München, April 2011.

[27] A. P. Dempster, N. M. Laird, D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39(1):1–38, 1977.

[28] JAMES project. Joint action for multimodal embodied social systems. `http://www.james-project.eu`, October 2011.

[29] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, April 1967.

[30] Zheng Wang, A. Peer, and M Buss. An hmm approach to realistic haptic human-robot interaction. *EuroHaptics conference, 2009 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, World Haptics 2009. Third Joint:374–379, 2009.

[31] Wikipedia. *Kinect.* September 15, 2011. `http://en.wikipedia.org/wiki/Kinect`.

[32] J. Yamato, J. Ohya, and K Ishii. Recognizing human action in time-sequential images using hidden markov model. *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR '92.*, 1992 IEEE Computer Society Conference:379–385, 1992.

[33] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. *The HTK Book (for HTK Version 3.4).* Cambridge University Engineering Department, September 17, 2011. `http://htk.eng.cam.ac.uk/`.