Freie Universität Berlin

Institut für Informatik

Bachelorarbeit

Sign Language Recognition with Kinect

Simon Lang

slang@zedat.fu-berlin.de

Supervisors:

Prof. Dr. Raúl Rojas

Prof. Dr. Marco Block-Berlitz

September 2011

Abstract

A framework for general gesture recognition is presented and tested with isolated signs of sign language. Other than common systems for sign language recognition, this framework makes use of *Kinect*, a depth camera developed by Microsoft and PrimeSense, which features easy extraction of important body parts. Recognition is done using hidden Markov models with a continuous observation density. The framework also offers an easy way of initializing and training new gestures or signs by performing them several times in front of the camera. Results show a recognition rate of $\geqslant 97\%$ for eight out of nine signs when they are trained by more than one person.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, diese Arbeit selbstständig und lediglich unter Benutzung der angegebenen Quellen und Hilfsmittel verfasst zu haben. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Berlin, den 27. September 2011

Simon Lang

Acknowledgements

First of all, I would like to thank my mentor Prof. Dr. Marco Block-Berlitz for supporting me and supervising my bachelor thesis. I would also like to thank my supervisor Prof. Dr. Raúl Rojas for his support.

Special thanks go to Fabian for performing and recording signs of German Sign Language, Mario for drawing the figure of the Urn and Ball Model, Anne for providing information on sign language reading material, and Achim for his general support. Special thanks also go to my sister Maren, my parents as well as all my friends for their support.

Contents

1 Motivation and Introduction

When Microsoft released Kinect in November 2010, it was mainly targeted at consumers owning a Microsoft Xbox 360 console. With the slogan "You are the controller", a controller-free experience for "games and entertainment" was advertised, allowing the user to interact with the system using gestures and speech [27, 28].

The device itself features an RGB camera, a depth sensor and a multiarray microphone, and is capable of tracking users' body movement [29]. The interest in it was high among developers and thus, shortly after its release an unofficial open source driver was introduced, followed by many Kinect-based projects and technical demos [36].

Since Kinect is able to track the user's full body, it seems natural to build a framework for sign language recognition. In sign languages, manual features are used along with facial expressions and different body postures in order to express words and grammatical features (p. 17 in [19]). Additionally, people and objects can be placed in front of a signing person and referred to during a conversation (pp. 57 and 61 in [19]).

Even though Microsoft stated that "Kinect that is shipping this [2010's] holiday will not support sign language", a demo video by the Center for Accessible Technology in Sign (CATS) shows how a few sign language sentences are recognized correctly on a system with limited vocabulary [30, 31]. The demo, however, does not support handshape recognition, and since sign language generally features different handshapes, similar signs cannot be distinguished.

The goal of this work is to create a framework for general gesture recognition that will be used to recognize signs of sign language (see figure 1.0.1). To achieve this, the framework will make use of hidden Markov models that allow training and recognition of isolated signs using only manual features.

Other systems capable of recognizing sign language rely on special gloves or ordinary RGB cameras that require well-lit environments. These systems have different advantages and disadvantages that will be reviewed, but this work will not focus on a comparison to those systems. Instead, an independent evaluation will show how it performs.

Finally, a conclusion will show the achievements of this work and what future work may follow in order to improve this framework and eliminate existing issues.

References have usually been put at the end of a paragraph and refer to the entire paragraph. When examples of signs are given, they are written in capital letters to indicate that the according translation of the word to a sign language is meant.

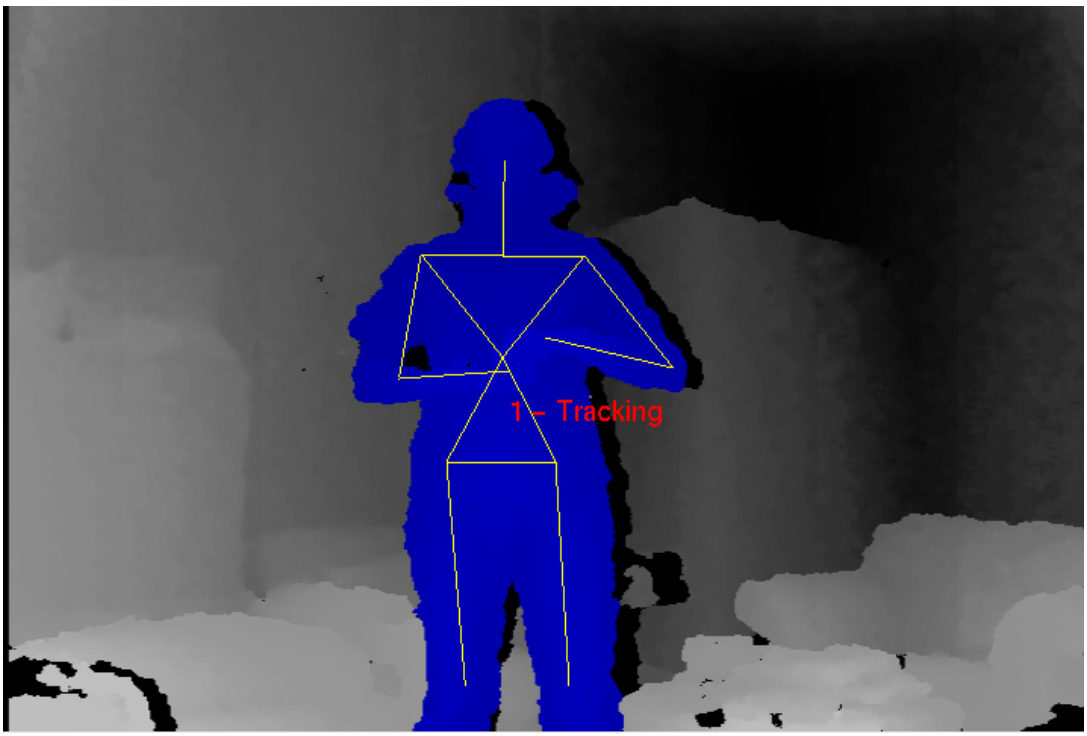

Figure 1.0.1: Sign language recognition with Kinect

2 Related Work

This part will summarize all basics that help create a better understanding of sign language and sign language recognition. Boyes Braem's "*Einführung in die Gebärdensprache und ihre Erforschung*" [19] as well as Rabiner's "*Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*" [20] are mainly used as references throughout the next sections, among other work more specifically related to sign language recognition.

2.1 Sign Language

The term *sign language* is similar to the term *language* in that there are many of both spread across different regions of the world. Just like languages, sign languages have evolved over a long time, they feature their own grammar and vocabulary, and thus they are considered real languages (pp. 9-10 in [19]).

The major difference between both is that sign languages are signed and languages are spoken, hence latter are also referred to as *spoken languages* in order to emphasize the difference. Due to the fact that no sense of hearing is required to understand sign language and no voice is required to produce it, it is the common type of language among deaf people (ibid.).

This section will give a short overview on the history of sign language and its research, followed by an introduction to the communication methods and grammatical features many sign languages have in common.

2.1.1 History

Up until the late 1960s, sign languages were generally not considered real languages. Instead, as Boyes Braem states, most linguists never seriously analyzed them and assumed they were just sets of gestures that could be loosely connected to express simple relations (ibid.).

Wilbur claims this was due to the negative attitudes towards sign language that generally existed, speaking of a "lack of understanding of the nature of language itself, and the related failure to separate language from speech". Another factor was the lack of traditional sign language ortography which has led to the use of sign names not including all necessary information for the appropriate signs. The example "John hit Bill" does not

convey that the sign HIT would start and end at different points when compared to "Bill hit John". Thus, many people compared these pseudo-translations to English, thinking sign language were incomplete and inferior (pp. 2-4 in [21]).

William C. Stokoe, a professor and English lecturer at Gallaudet University in the United States, concentrated on analyzing the American Sign Language (ASL) and was among the first to explore the communication methods of the deaf with up-to-date linguistic means. His work, however, was approached with scepticism, as even deaf people and hearing people with a sign language as their mother tongue considered their language underdeveloped and not a real language which could also be used for abstract communication that required higher education (pp. 10-11 in [19]).

Around the same time when Stokoe started his research on ASL, two other researchers, namely Ursula Bellugi and her husband Edward Klima, began a study on sign language during which they filmed ASL gestures of deaf people and analyzed them. Both never had any contact with deaf prior to their study and their aim was to analyze sign language from a neutral point of view without preconceived opinions. They came to the conclusion that ASL had its own grammar as well as most fundamental features that spoken languages have (p. 12 in [19]).

Boyes Braem summarizes the most important modern findings about sign language as follows:

- Sign language is a natural language that was not made up. Many deaf children acquire it as their mother tongue from other children at school or from their parents, which shows that the acquisition process is similar to that of spoken languages (p. 13 in [19]).
- Since it is a natural language, sign language is closely linked to the culture of the deaf, which it originates from. Thus, knowledge about the culture is necessary to fully understand sign language (ibid.).
- As mentioned above, there is not just a single sign language worldwide, but many national variants. These national variants can even have regional dialects (p. 14 in [19]).
- Unlike pantomime, sign language is not linked to iconic contents. Abstract thoughts can be expressed just as well as in spoken languages (ibid.).
- Furthermore, sign languages are not incomplete variants of languages spoken in the same region. Instead, they feature their own structure that can be completely different from that of the surrounding spoken languages (ibid.).

Even after the first results of modern sign language research were presented, they were not immediately accepted among both deaf and hearing people. For generations, the deaf were told their means of communication were not a real language and that they had to learn the spoken language of the majority. Many of them accepted and adopted this point of view and still thought of sign language as something inferior (pp. 11-12 in [19]).

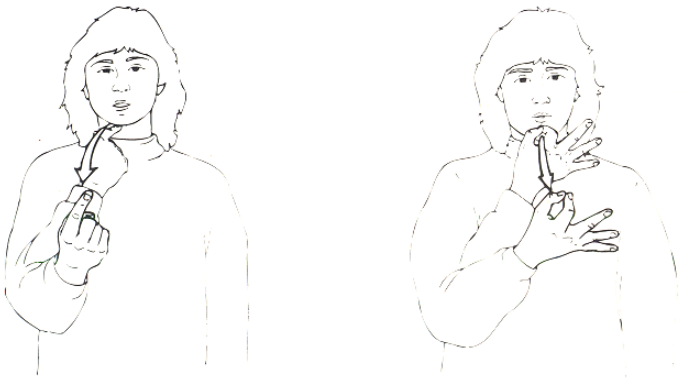

Figure 2.1.1: Signs for SAY (left) and ASK (right) (figure taken from [19], p. 19)

Hence, it took a while for sign languages to be recognized as real languages. Ever since the first modern studies by Stokoe and Klima and Bellugi, more and more research centers outside the United States have been established and science continued while the barrier to the modern view of sign language began to vanish (pp. 11-13 in [19]). In Germany, the local sign language (German Sign Language, Deutsche Gebärdensprache) has been made a legally recognized language in 2002 (§ 6 (1) in [12]). As a result, speech and hearing impaired people have the right to an interpreter when talking to public authorities (§ 9 (1) in [12]).

2.1.2 Sublexical Components

The means of communication in sign language are not limited to manual features, i. e. hands and arms, but also include non-manual signals such as facial and eye expressions, head and body posture, and mouth shape (p. 17 in [19]).

The manual components can be split into four parameters: *handshape*, *palm orientation*, *location*, and *movement*. A sign cannot be made with any handshape nor can any location be used. For example, in German-Swiss Sign Language, the manual signs for SAY (German: *sagen*) and ASK (*fragen*) are the same except for the handshape, see figure 2.1.1. In a similar way, signs exist that can only be distinguished by either orientation, location, or movement (pp. 18-26 in [19]).

Furthermore, not every possible handshape is available in each sign language, just as not every sound is available in each spoken language, e. g., two German pronounciations of "ch" are unavailable in English and the English "th" sounds are not used in German. As a result, a sign language can also be spoken with an accent, e. g., when signs from another sign language are borrowed that require a handshape unavailable in the actually used language. Instead, a known handshape replaces the unknown one (pp. 21-22 in [19]).

Although sign languages generally use different sets of handshapes, there are six shapes

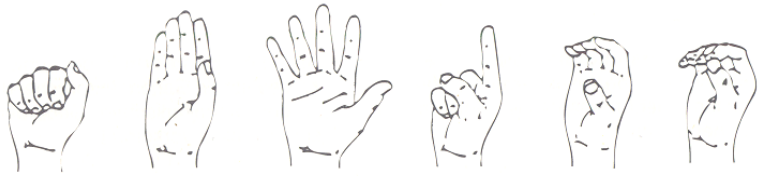

Figure 2.1.2: The six basic handshapes (figure taken from [19], p. 22)

Figure 2.1.3: Signing space (figure taken from [19], p. 23)

that were found to be used in all known sign languages, as shown in figure 2.1.2. Supposedly, these are the first to be acquired by deaf children (p. 22 in [19]).

Another important feature is the *signing space*. It spans around the head, the belly, and both arms and all signs are normally produced within that space (see figure 2.1.3). Signs can, however, also be made smaller or larger than usual, which represents the equivalent to whispering and screaming in spoken languages, respectively. Within the signing space, signs can be located directly at the body, near it, or at the other hand. A study by Liddell and Johnson claims there are 18 significant body locations for signs at the body, as shown in figure 2.1.4 (however, 20 are listed): both back and top of the head, forehead, temple, nose, cheek, ear, mouth, lip, lower jaw, chin, neck, shoulder, sternum, chest, upper body, upper arm, lower arm, belly, and leg (pp. 23-25 in [19]).

2.1.3 Sublexical Rules

The possible compositions of sign language components are limited by additional rules in each sign language. For example, in German the combination "pf" is valid whereas in English it is not. In a similar manner, several handshapes and other components can be combined and others cannot (p. 27 in [19]).

Figure 2.1.4: Significant body locations (figure taken from [19], p. 24)

Figure 2.1.5: A valid sign (left) and an invalid combination (right) (figure taken from [19], p. 28)

Many composition rules apply only to specific sign languages, but some can also be applied universally. Two of these rules have been found by Robbin Battison, namely the *symmetry condition* and the *dominance condition*. The symmetry condition implies that if both hands move at the same time during a sign, they must use the same handshape. Figure 2.1.5 shows a valid sign next to a combination that violates this rule [22].

The dominance condition implies that if both hands use different handshapes, the dominant hand is moving and the passive hand must stand still. Additionally, the passive hand must feature one of the six basic handshapes shown in figure 2.1.2 (ibid.).

2.1.4 Grammar

Sign languages feature the same universal linguistic functions that all other natural languages do. Although there exist features in some spoken languages that cannot be found

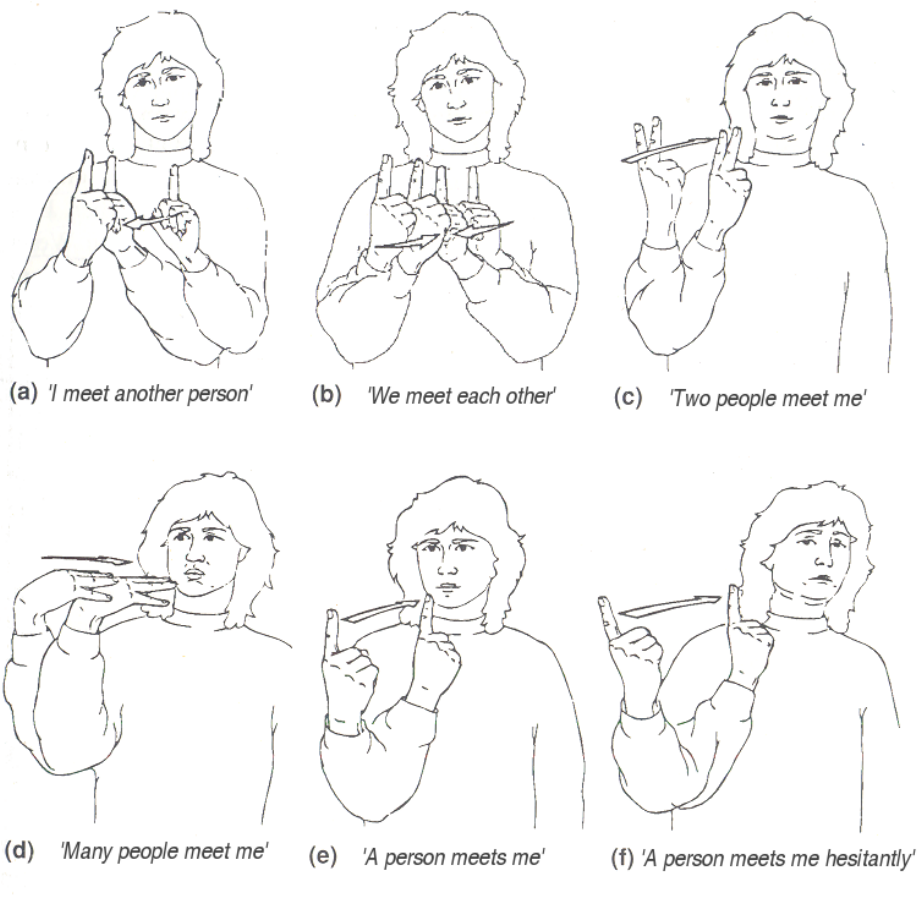

Figure 2.1.6: Modification of the sign MEET (translated version of the original figure in [19], p. 51)

in sign languages, such as articles (e. g. "*the*") or copulae (e. g. "I *am* interested."), the lack of these features is not specific to sign languages as they cannot be found in some spoken languages, such as Chinese, either. Yet, features such as the ability to reference a person (e. g. *I*, *she*, *you*) or to indicate a tense (present, future, past) are indeed available in all sign languages and all spoken languages (p. 45 in [19]).

A study by Bellugi and Fischer showed that native speakers of both English and a sign language could produce about half as much signs per second as words per second. However, the amount of information per second was about the same, which is due to the fact that signs contain lots of information which can be conveyed concurrently, while speech delivers information sequentially. Figure 2.1.6 shows several variations of the sign MEET. Composed signs or pre- and suffixes, on the other hand, are very rarely used in sign languages (pp. 46-52 in [19]).

An important component of expressing grammatical features is the location of signs. Sign

language grammar includes four aspects that depend on the use of different executing locations:

- Marking the origin and destination of an action
- Pronominal reference
- Marking subject and accusative or dative object by means of the verb
- Indicating tense

Origin and destination, referencing objects and people

Many verbs are inflected in order to convey additional information. As figure 2.1.7 shows, the verb GO is inflected to indicate that the signing person will go from point A (STORE) to point B (SCHOOL), i. e., the GO sign starts where STORE was signed and ends where SCHOOL was signed. The figure also illustrates how objects can be placed in front of a signing person (pp. 55-58 in [19]).

If STORE or SCHOOL were used again later during the conversation, they could be referenced by pointing at them using the INDEX sign, without the need to perform their actual signs again. Similarly, an absent person that is referred to can be placed in front of a signer, while present people are referenced by pointing at them directly (ibid).

Indicating tense

Different executing locations are also used to indicate tense. In English and other languages, expressions like "tomorrow", "last summer", or "in five years" are used, and similar signs can also be found in sign languages.

They generally follow specific rules, for example in Western European sign languages an imaginary line is drawn through one's body like a path that has been walked. All signs in front of the body refer to something in the future, while signs near the body indicate present tense and backward-oriented signs are linked to a past event (pp. 69-70 in [19]).

Classifying signs

Some signs are modified in order to show which type of subject or object they refer to. They can classify things by several properties such as shape, size or weight. Figure 2.1.8 illustrates how the verb EAT is modified to classify a small and rotund object as opposed to something small and angular (pp. 75-90 in [19]).

A similar feature is used in Japanese where different suffixes of counter words classify what type of object is referenced, e. g. different words for "*two*" would be used when referring to small animals in contrast to human beings (p. 76 in [11]).

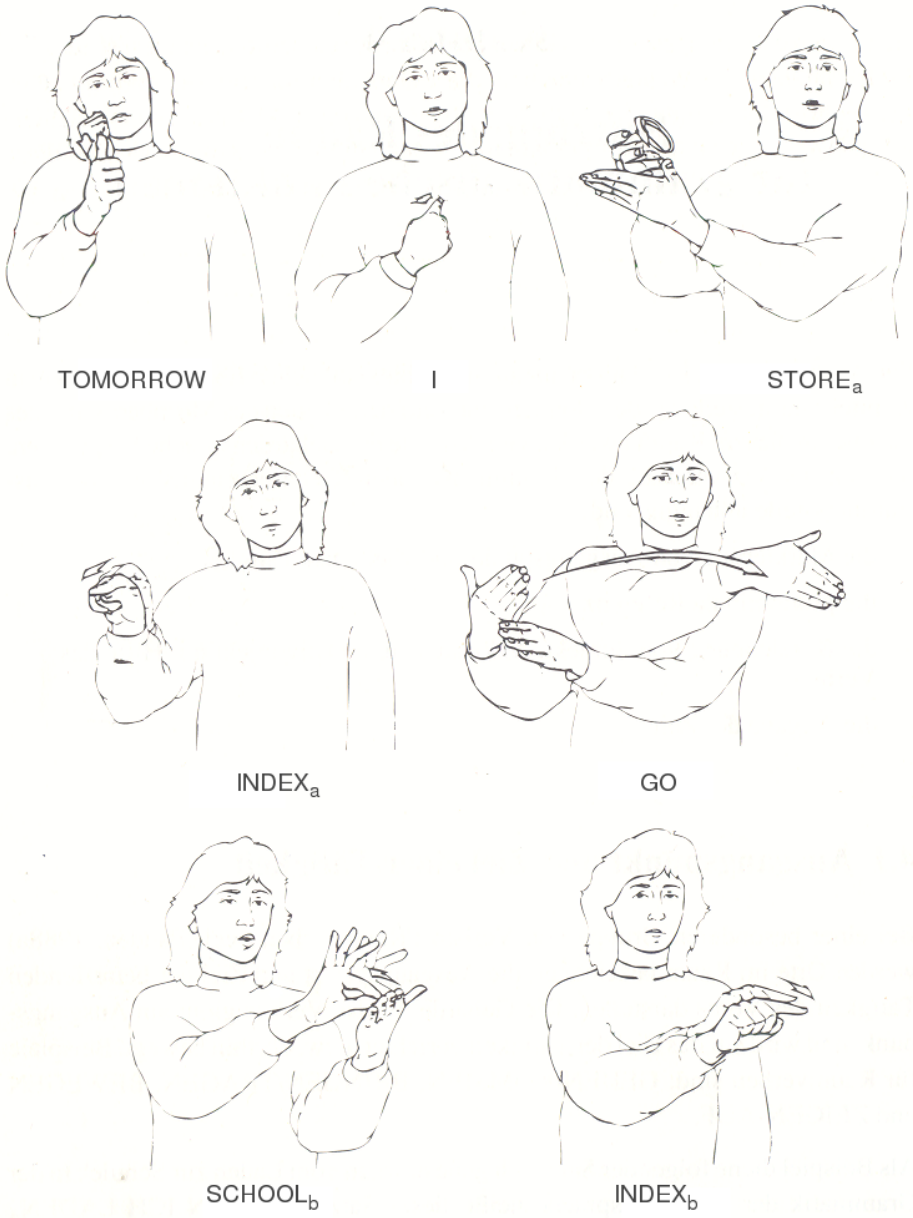

Figure 2.1.7: Inflecting verbs and placing objects in the signing space (translated version of the original figure in [19], p. 56)

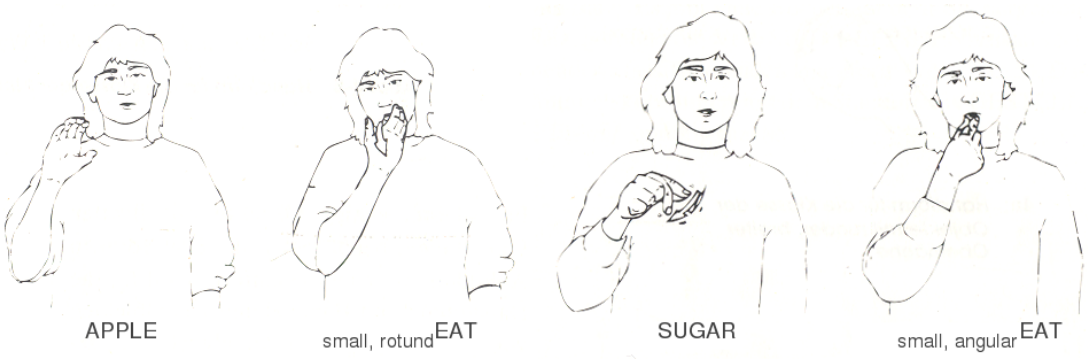

Figure 2.1.8: Modifications of the verb EAT (translated version of the original figure in [19, pp. 81-82])

Non-manual features

So far, only manual components have been discussed, but the non-manual components are also used to express important grammatical features. These include, but are not limited to,

- non-manually expressed adjectives and adverbs,
- different sentence types such as negation, affirmation, question, relative and conditional clauses, and
- mouth shapes (pp. 97-99 in [19]).

To further describe a noun or verb, non-manual adjectives or adverbs can be added while signing. These include facial and eye expression, head and body posture, and mouth shape. An example in figure 2.1.9 shows how the sentence "*The car drives by*" is modified with an adverb to mean "*The car drives by closely*". The modification *close/closely* is done by tilting the head towards the shoulder and moving the mouth corner in the same direction (pp. 99-104 in [19]).

Questions that require *yes* or *no* as an answer are produced by signing exactly the same sentence as a statement and additionally tilting the head forward with eyes wide open and eyebrows raised. Other questions using question words (*how*, *when*, *why*, *who*, *what*, *where*) involve signing the according question word first and keeping a specific facial expression throughout the entire question. Negating or affirming a question or statement can be done by either signing the appropriate words for *no* or *yes,* or by repeating the sentence while shaking the head or nodding, respectively (pp. 105-108 in [19]).

Finally, signs can be accompanied by mouth shapes. These can have, among others, a *distinguishing role* (e. g. in some German-Swiss Sign Language dialects, BROTHER and SISTER are signed the same way but with different mouth shapes), a *specifying role* or an *emphasizing role*. Sometimes, mouth shapes are isolated and not accompanied by signs. This can happen if an according sign does not exist or is not known, e. g. the name *Susanna* (pp. 117-118 in [19]).

Figure 2.1.9: Non-manual adverb "*closely*" (figure taken from [19], p. 102)

2.2 Kinect

Kinect, code-named Project Natal, was developed by Microsoft and PrimeSense and released in November 2010. It features an RGB camera along with a microphone array and a depth sensor using an infrared projector, and is thus capable of tracking a user's full body independently from lighting conditions [33, 32, 29].

Kinect is primarily targeted at owners of a Microsoft Xbox 360 console and advertised with the slogan "You are the controller", promising a video game and entertainment experience without the need for gamepads or other devices for interaction [27].

Beside its application on Xbox 360, several drivers exist that allow Kinect to be used on PC and Mac. They have different advantages and disadvantages and will be reviewed in this section.

2.2.1 Existing Frameworks

Soon after Kinect was released, a company called Adafruit Industries that sells open source hardware electronics, offered a bounty of 1,000 US dollars for the first who would release "completely documented" drivers for Kinect "under an open source license" [34]. The bounty was later increased to 2,000 and then to 3,000 US dollars when Microsoft indirectly replied to the offer, stating that the company "does not condone the modification of its products" and would "work closely with law enforcement [...] to keep Kinect tamper-resistant" [35].

Less than a week later, Adafruit announced the winner of the contest and paid the promised bounty. Additionally, 2,000 US dollars were donated to the Electronic Frontier

Foundation (EFF) for defending digital rights and the "right [...] to do things like this project", in response to Microsoft's statement [36].

In an interview at NPR (National Public Radio) in the United States, however, a Microsoft spokesperson said that "Kinect was not actually hacked" since the driver "essentially opens the USB connection, which [Microsoft] didn't protect by design", and neither Adafruit nor the contest winner would be sued for that [37]. It was later revealed that Johnny Lee, former employee at Microsoft, was the one who initiated the contest while still with the company, and asked Adafruit for help after his approach to have Microsoft develop a driver failed [38].

The driver that emerged from this contest was named *libfreenect* and initially made available for Linux only under the terms of either the Apache 2.0 license or optionally version 2 of the GNU General Public License. It is still under development by the OpenKinect community, and high-level features such as skeletal tracking are not yet offered, but it has already been ported to Microsoft Windows and Mac OS X. Its current capabilities include displaying Kinect's RGB and depth streams as well as using the built-in microphone array and motorized tilt mechanism. Bindings are available for several programming languages such as Python, C, C++, C# and Java [39, 40].

Another driver called *OpenNI* was released in December 2010 by PrimeSense, the manufacturer of Kinect's camera technology. It includes a feature-rich open source framework licensed under the GNU Lesser General Public License, version 3, and can be combined with closed source middleware called *NITE* for skeletal tracking and recognition of hand gestures [41, 42]. An example of skeletal tracking is shown in figure 2.2.1.

In June 2011, Microsoft released an official SDK for Microsoft Windows. The *Kinect for Windows SDK Beta* is for personal and academic, i. e. non-commercial, use only, and features a rich API that offers similar features as OpenNI. It provides access to raw sensor streams and skeletal tracking, and features bindings to several programming languages such as C++, C# and Visual Basic. Microsoft plans to release the SDK in a commercial version at a later date [45].

While skeletal tracking is supported by both OpenNI and the Kinect for Windows SDK, platform-independency is offered by libfreenect and OpenNI. Currently, the latter is often used in demos and applications that need both, since it combines the advantages of skeletal tracking and platform-independency.

2.2.2 Limitations and Future Development

Although the two frameworks by Microsoft and PrimeSense offer skeletal tracking, both do not support recognition of single fingers, and thus cannot distinguish handshapes. Evoluce AG, a manufacturer of multitouch displays, announced a *Multi-Gesture SDK* for Kinect and similar depth-sensing cameras to overcome this issue. The SDK will be based on OpenNI and is said to support finger gestures, among other features [46].

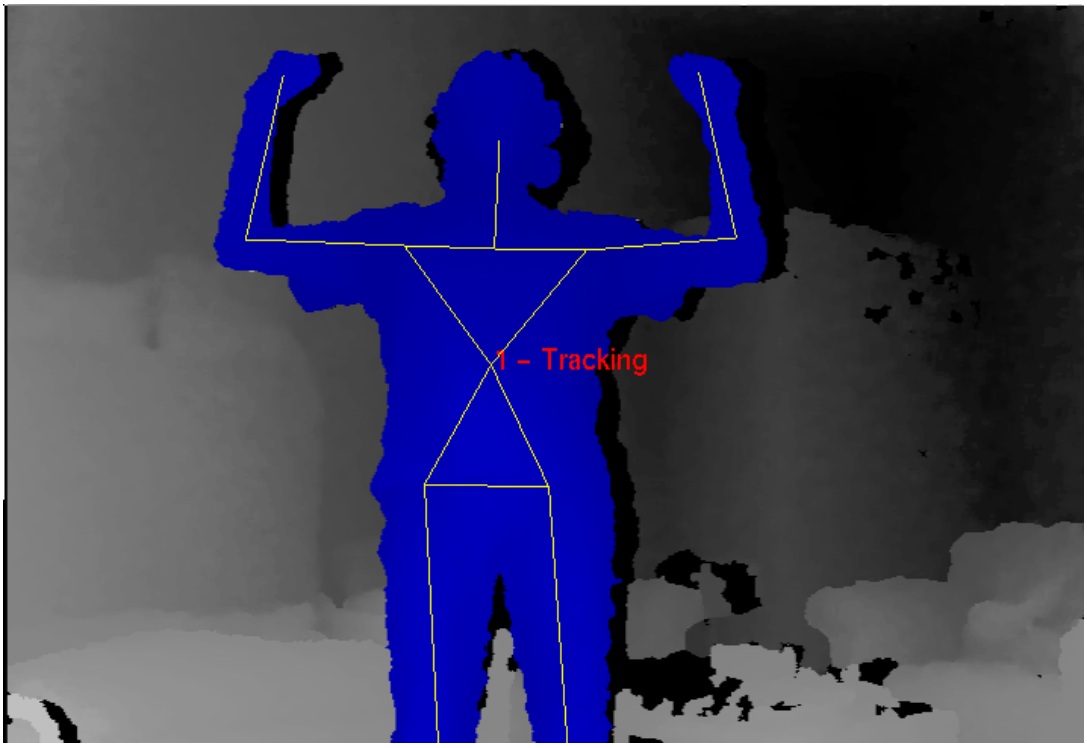

Figure 2.2.1: Skeletal tracking with OpenNI

A video by Dr. Natheer Khasawneh at Jordan University of Science and Technology shows a Kinect demo for Arabic Sign Language recognition that is capable of tracking the user's fingers [47]. However, the application is not publicly available.

In addition to the basic libfreenect driver the OpenKinect community provides, an *OpenKinect Analysis Library* is planned. Feature suggestions include hand tracking and skeletal tracking [40].

Until these technologies are made available, the existing frameworks feature customizability and can be extended by manually processing the camera's depth stream.

2.3 Hidden Markov Models

Hidden Markov models (HMMs) are a type of stochastic model related to finite state machines (FSMs) and are often used in recognition of speech, handwriting, and also sign language. This section will first describe basic Markov chains and how they can be extended to HMMs, and then give an overview of the three basic problems that HMMs come along with.

2.3.1 Markov Chains

A *Markov chain*, also called *discrete Markov process* or *observable Markov model*, can be seen as a finite state machine with N states, $S_1, S_2, \cdots, S_n$, being in exactly one of these states at any time. At regular intervals, the state will change according to a probability associated with the current state, either to a different state or back to the same state. The time instants where state changes occur will be referenced as $t = 1, 2, 3, \cdots$, whereas q_t will refer to the state at time t [20, 15].

An important restriction for Markov chains is that the probability of changing to a specific state only depends on the current state and not its predecessor states, so the matrix A of state transition probabilities can be written as

$$a_{ij} = P[q_{t+1} = S_j \,|\, q_t = S_i], \quad 1 \leqslant i, j \leqslant N \tag{2.3.1}$$

where the normal stochastic constraints should be satisfied, i. e.

$$a_{ij} \geqslant 0 \tag{2.3.2}$$

$$\text{and } \sum_{j=1}^{N} a_{ij} = 1. \tag{2.3.3}$$

Rabiner [20] brings up an example with a 3-state weather model as shown in figure 2.3.1, where the weather is one of the following: rain (state S_1), cloudy (state S_2), or sunny

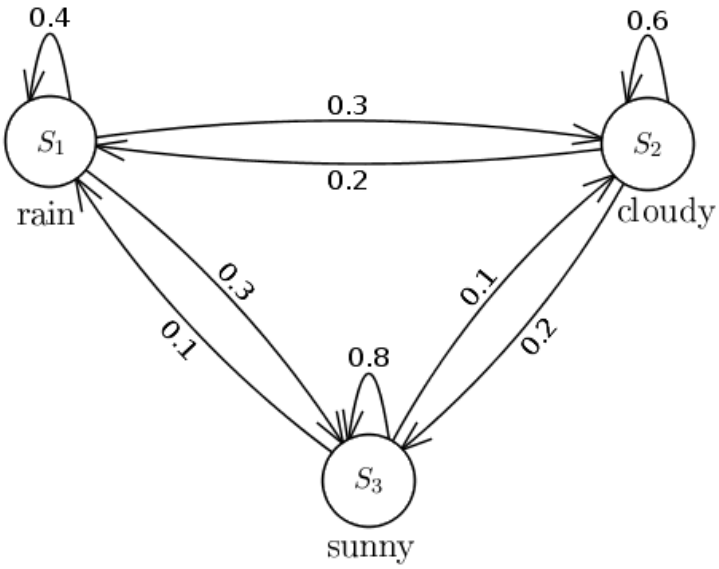

Figure 2.3.1: 3-state weather model (according to the example in [20])

(state S_3). The state transition probability matrix could be

$$A = \{a_{ij}\} = \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{pmatrix}.$$

Given this model, for any observation sequence O a probability can be calculated. Let state changes occur between each two days, making q_t refer to the weather on day t. For instance, given that the first day is sunny, the weather observation of six successive days could be $O = \{S_3, S_3, S_1, S_3, S_2, S_2\}$ $(t = 1, 2, \cdots, 6)$, and the probability for O could be calculated as follows,

$$\begin{aligned} P(O \,|\, \text{Model}) &= P[S_3, S_3, S_1, S_3, S_2, S_2 \,|\, \text{Model}] \\ &= P[S_3] \cdot P[S_3|S_3] \cdot P[S_1|S_3] \cdot P[S_3|S_1] \cdot P[S_2|S_3] \cdot P[S_2|S_2] \\ &= \pi_3 \cdot a_{33} \cdot a_{31} \cdot a_{13} \cdot a_{32} \cdot a_{22} \\ &= 1 \cdot 0.8 \cdot 0.1 \cdot 0.3 \cdot 0.1 \cdot 0.6 \\ &= 1.44 \cdot 10^{-3}, \end{aligned}$$

where π_3 is the initial probability of the weather being sunny, which was taken for granted. The general notation for initial state probabilities is

$$\pi_i = P[q_1 = S_i] \quad , 1 \leqslant i \leqslant N. \tag{2.3.4}$$

A strong limitation of Markov chains is that they rely on observable events (such as *sun*, *rain*, *clouds*) and each state corresponds to exactly one such event, making it possible to

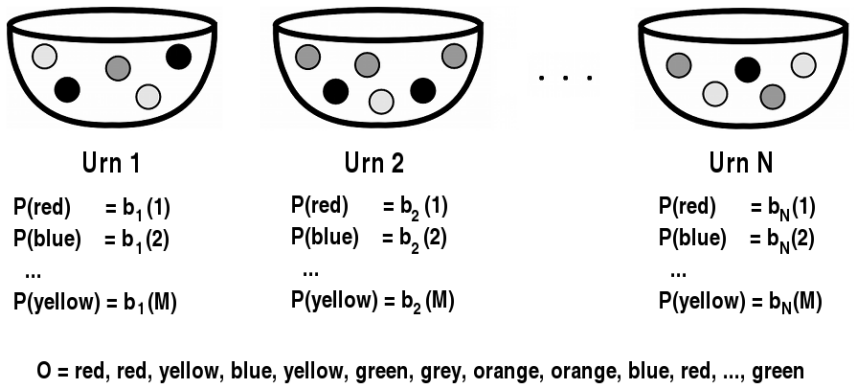

Figure 2.3.2: Urn and Ball Model (figure drawn by Mario Dederichs according to the example in [20])

conclude the current state from an observation. In order to be applied to problems such as recognition of speech, handwriting or sign language, however, a less restrictive model is needed (ibid.).

2.3.2 Extending Markov Chains

In HMMs, each state produces one out of a set of observable events according to a probablistic function associated with that state, rather than producing the same fixed event each time. Thus, an observation does no longer imply the current state, it can only hint at it – the process of state transition is now *hidden* [20, 15].

Another example Rabiner [20] brings up, was introduced by Jack Ferguson et al. and is called *The Urn and Ball Model*, shown in figure 2.3.2. There are N large urns in a room that hold a large number of colored balls each, and each ball has one out of M distinct colors. The process for generating observations involves a genie who picks an initial urn according to some random procedure. He or she randomly chooses a ball from that urn, and its color is noted down for observation. The ball is put back in the urn and a new urn is selected according to a stochastic process associated with the current urn. This ball choosing and urn selection process is now repeated and will generate a finite observation sequence of colors.

When modelling an HMM for this example, the N large urns correspond to the N states. The process of picking an initial urn i marks the initial state probability π_i (see equation 2.3.4), and selecting a new urn corresponds to the state transition probability a_{ij} as defined in equation 2.3.1, given that the genie switches from urn i to urn j.

Since the M observation symbols are, unlike in Markov chains, no longer strictly linked to a specific state each, they have to be considered separately. They are denoted as

$V = \{v_1, v_2, \cdots, v_M\}$. Consider the above example with $M = 4$, then V could be $V = \{\text{red, green, blue, yellow}\}$ for instance.

Additionally, a set of probability distributions for observation symbols in each state is needed, which is denoted as $B = \{b_j(k)\}$, where

$$b_j(k) = P[v_k \text{ at } t \,|\, q_t = S_j], \quad \begin{aligned} 1 &\leqslant j \leqslant N \\ 1 &\leqslant k \leqslant M. \end{aligned} \tag{2.3.5}$$

Following the above example, consider there are four balls in urn 1, one of each color, and ten balls in urn 2, seven of them red, two green and one blue. The probability distributions would be

$$b_1(k) = 0.25 \quad , k = 1, 2, 3, 4$$
$$b_2(k) = \begin{cases} 0.7 & , k = 1 \\ 0.2 & , k = 2 \\ 0.1 & , k = 3 \\ 0 & , k = 4 \end{cases}.$$

Finally, an observation sequence can be referenced as

$$O = O_1 O_2 \cdots O_T, \tag{2.3.6}$$

where each O_t $(t = 1, \cdots, T)$ is a symbol from V, i. e. an observation sequence in the above example could be $O = v_1 v_1 v_1 v_4 v_2 v_3 v_2$ (red, red, red, yellow, green, blue, green). Observation sequences are generated as follows [20]:

1. Use the initial state distribution π to choose the initial state $q_1 = S_i$.
2. Set $t = 1$.
3. Use the symbol probability distribution $b_i(k)$ in state S_i to choose the current observation symbol $O_t = v_k$.
4. Choose a new state $q_{t+1} = S_j$ using the state transition probability distribution a_{ij} of state $q_t = S_i$.
5. Set $t = t + 1$ and return to step 3 if $t < T$, else terminate.

Using the three probability distributions, i. e. those for state transitioning, symbol choosing, and initial state choosing, an HMM can be denoted as

$$\lambda = (A, B, \pi). \tag{2.3.7}$$

2.3.3 The Three Basic Problems

Given this compact notation, Rabiner summarizes the three basic problems of HMMs as follows [20]:

1. Given a model $\lambda = (A, B, \pi)$ and an observation sequence $O = O_1 O_2 \cdots O_T$, determine the probability of O being generated when λ is used, i. e. efficiently calculate $P(O|\lambda)$.
2. Determine the *optimal* state sequence $Q = q_1 q_2 \cdots q_T$, i. e. the one that is most likely to be traversed, given an obversation sequence O and a model λ.
3. Determine how to adjust the parameters of λ in order to maximize $P(O|\lambda)$.

These problems are also known as the *evaluation problem*, the *decoding problem*, and the *optimizing problem*, as explained detailedly below [20, 15].

Solution to the First Problem

This problem can be seen as determining how well the given model corresponds to a given observation sequence. This is useful when choosing between one out of several competing models, e. g. in sign language recognition where each word matches an HMM, the HMM λ_i with the highest $P(O|\lambda_i)$ would determine the recognized word [20, 15].

The easiest way of solving the problem would involve enumerating all possible state sequences of the same length T as the observation sequence, calculating the probability for each and then summing up all these probabilites to determine $P(O|\lambda)$. However, this calculation method would be very inefficient, and thus the more efficient *Forward-Backward Procedure* is used [20].

It defines the forward variable $\alpha_t(i)$ as the probability of a partial observation sequence $O_1 O_2 \cdots O_t$ up to $t < T$ with the model being in state S_i at time t,

$$\alpha_t(i) = P(O_1 O_2 \cdots O_t, q_t = S_i \,|\, \lambda). \tag{2.3.8}$$

The problem can now be solved inductively through

1. Initialization,

$$\alpha_1(i) = \pi_i b_i(O_1) \quad , 1 \leqslant i \leqslant N \tag{2.3.9}$$

2. Induction,

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) \cdot a_{ij}\right] \cdot b_j(O_{t+1}) \quad , 1 \leqslant t \leqslant T-1$$
$$1 \leqslant j \leqslant N \tag{2.3.10}$$

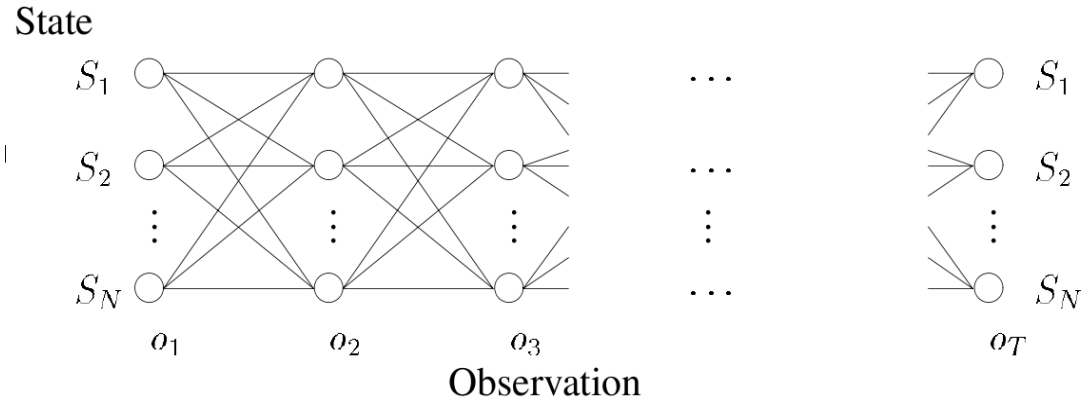

Figure 2.3.3: Forward-Backward Procedure (original figure in [13])

3. Termination,

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i). \tag{2.3.11}$$

Figure 2.3.3 outlines how the induction step works. In each step, the probability of each previous partial observation sequence is multiplied with the associated state transition probability. The products are summed up and multiplied with the symbol probability for the current observation symbol. That way, at each step all previous partial observation sequence probabilities merge into each new such probability. This is possible because the process of choosing a next state only depends on the current state and not any predecessor states. Finally, the termination sums up the results of the last step (ibid.).

In a similar way, a backward variable $\beta_t(i)$ can be defined as the probability of a partial observation sequence $O_{t+1}O_{t+2}\cdots O_T$ from $t+1$ to T with the model being in state S_i at time t:

$$\beta_t(i) = P(O_{t+1}O_{t+2}\cdots O_T \,|\, q_t = S_i, \lambda) \tag{2.3.12}$$

Just as with the forward variable, the problem can also be solved using the backward variable:

1. Initialization,

$$\beta_T = 1 \quad , 1 \leqslant i \leqslant N \tag{2.3.13}$$

2. Induction,

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(j) \quad , t = T-1, T-2, \cdots, 1$$
$$1 \leqslant i \leqslant N \tag{2.3.14}$$

3. Termination,

$$P(O|\lambda) = \sum_{j=1}^{N} \pi_j \cdot b_j(O_1) \cdot \beta_1(j). \tag{2.3.15}$$

In order to calculate $P(O|\lambda)$, only one of these methods is necessary. However, for attempting to solve the third problem of HMMs, both the forward and the backward variable are needed (ibid.).

Solution to the Second Problem

The second problem tries to decode the hidden part of the HMM, i. e. the state sequence $Q = q_1 q_2 \cdots q_T$ given the model and an observation sequence $O = O_1 O_2 \cdots O_T$. The state sequence cannot be determined with absolute certainty, so the problem is solved as good as possible using the *Viterbi Algorithm*.

The Viterbi algorithm defines $\delta_t(i)$ as the highest probability along a single path up to time $t \leqslant T$. Additionally, a variable that keeps track of the state that was transitioned to has to be defined, called $\psi_t(i)$. This variable is needed because the desired result is the actual state sequence with the highest probability, and not the highest probability itself. The variables are defined inductively as follows (ibid.):

1. Initialization:

$$\delta_1(t) = \pi_i \cdot b_i(O_1) \quad , 1 \leqslant i \leqslant N \tag{2.3.16}$$

$$\psi_1(i) = 0 \tag{2.3.17}$$

2. Induction:

$$\delta_t(j) = \max_{1 \leqslant i \leqslant N} [\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(O_t) \quad , 2 \leqslant t \leqslant T \quad 1 \leqslant j \leqslant N \tag{2.3.18}$$

$$\psi_t(j) = \operatorname*{argmax}_{1 \leqslant i \leqslant N} [\delta_{t-1}(i) \cdot a_{ij}] \quad , 2 \leqslant t \leqslant T \quad 1 \leqslant j \leqslant N \tag{2.3.19}$$

3. Termination:

$$p^* = \max_{1 \leqslant i \leqslant N} [\delta_T(i)] \tag{2.3.20}$$

$$q_T^* = \operatorname*{argmax}_{1 \leqslant i \leqslant N} [\delta_T(i)] \tag{2.3.21}$$

4. State sequence backtracking:

$$q_t* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \cdots, 1 \tag{2.3.22}$$

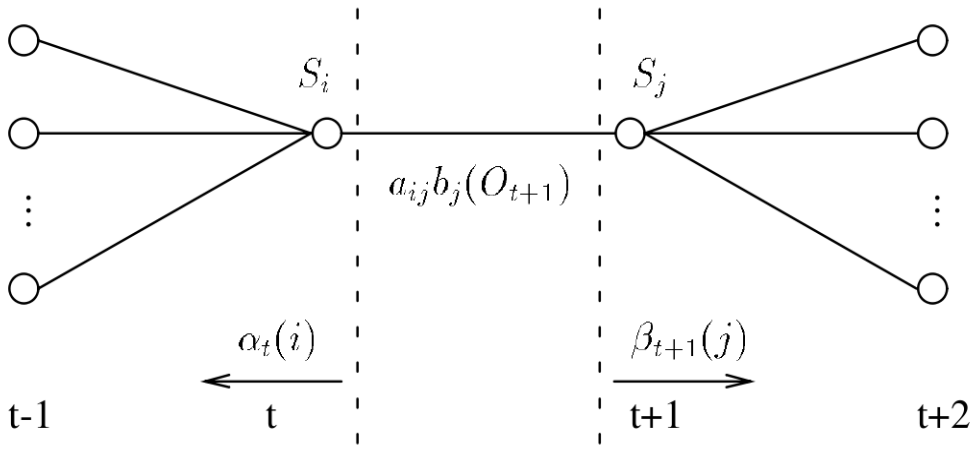

Figure 2.3.4: Illustration of $\xi_t(i,j)$ (figure taken from [13])

The Viterbi algorithm is similar to the calculation of the forward variable in the Forward-Backward algorithm, except the maximum probability of the previous iteration is used rather than the sum of previous probabilities. The actual state sequence can be extracted using $\psi_t(j)$ which depends on the calculations of $\delta_{t-1}(i)$ (ibid.).

Solution to the Third Problem

The third problem corresponds to training HMMs in order to make them recognize previously unknown data. There is no known way of analytically solving this problem, i. e. maximizing $P(O|\lambda)$. However, it can be maximized locally using iterative algorithms, e. g. the Baum-Welch method [20, 15].

First, $\xi_t(i,j)$ needs to be defined as the probability of being in state S_i at time t and in state $S_j$at time $t+1$, given an observation sequence O and a model λ, as illustrated in figure 2.3.4, i. e.

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j|O,\lambda). \tag{2.3.23}$$

Using the forward and backward variables from 2.3.10 and 2.3.14, $\xi_t(i,j)$ can be written as follows [20],

$$\begin{aligned}\xi_t(i,j) &= \frac{\alpha_t(i)\cdot a_{ij}\cdot b_j(O_{t+1})\cdot\beta_{t+1}(j)}{P(O|\lambda)} \\ &= \frac{\alpha_t(i)\cdot a_{ij}\cdot b_j(O_{t+1})\cdot\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)\cdot a_{ij}\cdot b_j(O_{t+1})\cdot\beta_{t+1}(j)}.\end{aligned} \tag{2.3.24}$$

Another formular that is needed defines $\gamma_t(i)$ as the probability of being in state S_i at time t, given an observation sequence O and a model λ, i. e.

$$\gamma_t(i) = P(q_t = S_i|O, \lambda). \tag{2.3.25}$$

This can be expressed as

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i, j). \tag{2.3.26}$$

If $\gamma_t(i)$ is summed over time t, the result is the expected number of transitions from state S_i. In a similar manner, summing $\xi_t(i, j)$ over time results in the expected number of transitions from S_i to S_j (ibid.):

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from } S_i \tag{2.3.27}$$

$$\sum_{t=1}^{T-1} \xi_t(i, j) = \text{expected number of transitions from } S_i \text{ to } S_j. \tag{2.3.28}$$

Given these and the previous formulas, as well as an observation sequence O, the parameters of an HMM can be reestimated,

$$\overline{\pi}_i = \text{expected number of times in state } S_i \text{ at time } (t = 1) = \gamma_1(i), \tag{2.3.29}$$

$$\begin{aligned} \overline{a}_{ij} &= \frac{\text{expected number of transitions from state } S_j \text{ to state } S_j}{\text{expected number of transitions from state } S_i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \end{aligned} \tag{2.3.30}$$

$$\begin{aligned} \overline{b}_j(k) &= \frac{\text{expected number of times in state } S_j \text{ and observing symbol } v_k}{\text{expected number of times in state } S_j} \\ &= \frac{\sum_{t \in [1,T] \wedge O_t = v_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}. \end{aligned} \tag{2.3.31}$$

This new HMM is called $\overline{\lambda}$. Baum and his colleages have proven that the probability of the sequence used for reestimating, given the new model $\overline{\lambda}$, is either higher than or equal to the probabiliy of the sequence, given the old model, i. e. $P(O|\overline{\lambda}) \geqslant P(O|\lambda)$. If both probabilities are equal, then $\lambda = \overline{\lambda}$ (ibid.).

2.3.4 Types of Models

The type of HMMs assumed for all above formulas and theorems is the ergodic model (shown in figure 2.3.5 on the left). This model is the most general type where all states

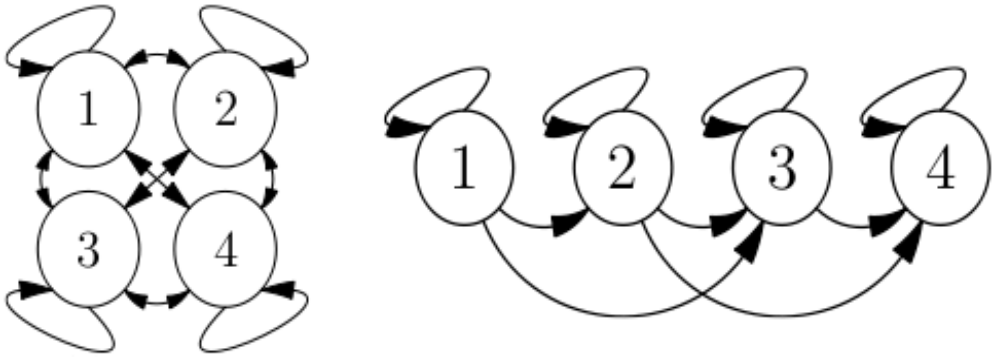

Figure 2.3.5: Ergodic model (l.) and LR-model (original figures in [4])

are interconnected and the initial probability for each state can be any value (as long as the stochastic constraints are satisfied) [20].

Additional restrictions can be imposed on this type, which results in more specific models such as the LR-model (figure 2.3.5, on the right). This type can be of interest in language recognition since it can represent how a signal changes over time. An example transition probability matrix could be

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

The *reach* R of such a model is defined as the number of states that a state S_i can reach (excluding S_i itself) [20, 7, 4]. In the above example, the reach would be 2. These properties can be formulated as

$$a_{ij} = 0 \qquad , j < i \tag{2.3.32}$$
$$a_{ij} = 0 \qquad , j > i + R. \tag{2.3.33}$$

Further constraints are imposed on the initial state probabilities, such that

$$\pi_i = \begin{cases} 1 & , i = 1 \\ 0 & , i > 1 \end{cases} \tag{2.3.34}$$

There are more types of models that will not be discussed as they are not as important for this thesis as ergodic and LR-models.

2.3.5 Continuous Observation Density

A downside of finite observation symbol alphabets, as used before, is that in many applications of interest the observations are continuous vectors, e. g. vectors of real numbers.

A possible solution to this problem lies in quantization, such that a symbol from the alphabet close to the actual observation is used. However, the signal might be severely degraded due to such a process, and thus a more convenient approach would be to use continuous observation densities with HMMs, i. e. an infinite alphabet of observation symbols [20, 15, 8].

Due to this infinite alphabet, *continuous density HMMs* (CD-HMMs) cannot use a discrete probability density in each state. Instead, they make use of a probability density function (pdf) which is usually a mixture of weighted logarithmically concave functions, e. g. Gaussian distributions. Using a mixture model with M components, for an observation symbol vector O the probability distribution is expressed as follows [20, 8],

$$b_j(O) = \sum_{m=1}^{M} c_{jm} \cdot f[O, \mu_{jm}, \Sigma_{jm}] \quad , 1 \leqslant j \leqslant N. \tag{2.3.35}$$

The logarithmically concave function f has a mean vector μ_{jm} and a covariance matrix Σ_{jm} as additional arguments, both depending on the state S_j and current mixture m. c_{jm} are the weighting coefficients, which satisfy the stochastic constraints,

$$\sum_{m=1}^{M} c_{jm} = 1 \qquad\qquad , 1 \leqslant j \leqslant N \tag{2.3.36}$$

$$c_{jm} \geqslant 0 \quad , 1 \leqslant j \leqslant N, 1 \leqslant m \leqslant M \tag{2.3.37}$$

since the pdf's area should be 1, i. e. it should be normalized [20].

The parameters and weighting coefficients can also be reestimated to improve $P(O|\lambda)$, similar to equations 2.3.29-2.3.31, as follows,

$$\overline{c}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \gamma_t(j,k)}, \tag{2.3.38}$$

$$\overline{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot O_t}{\sum_{t=1}^{T} \gamma_t(j,k)}, \tag{2.3.39}$$

$$\overline{\Sigma}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k) \cdot (O_t - \mu_{jk}) \otimes (O_t - \mu_{jk})^{\mathsf{T}}}{\sum_{t=1}^{T} \gamma_t(j,k)}, \tag{2.3.40}$$

where $()^{\mathsf{T}}$ marks vector transposing, $\otimes$ the dyadic product, and $\gamma_t(j,k)$ the probability of being in state S_j with the k^{th} mixture component for O_t at time t (ibid.),

$$\gamma_t(j,k) = \left[\frac{\alpha_t(j) \cdot \beta_t(j)}{\sum_{j=1}^{N} \alpha_t(j) \cdot \beta_t(j)} \right] \cdot \left[\frac{c_{jk} \cdot f(O_t, \mu_{jk}, \Sigma_{jk})}{\sum_{m=1}^{M} c_{jm} \cdot f(O_t, \mu_{jm}, \Sigma_{jm})} \right]. \tag{2.3.41}$$

This term equals $\gamma_t(j)$ from 2.3.26 if the mixture is simple, i. e. $M = 1$ (and thus $k = 1$) (ibid.).

2.3.6 Scaling

When implementing HMMs, more issues emerge that need to be solved. One of these issues is arithmetic underflow during probability calculations. An example would be the forward variable $\alpha_t(i)$, which is a sum of terms of the following form,

$$\left(\prod_{s=1}^{t-1} a_{q_s} a_{q_{s+1}} \prod_{s=1}^{t} b_{q_s}(O_s)\right) \qquad , q_t = S_i$$

where in general the a and b terms are all significantly less than 1. For large values of t, this will result in result in very small numbers, eventually exceeding the precision range of a computer (the backward variable β shows similar behavior). Thus, the α and β variables have to be scaled in order to prevent underflow [20].

The goal is to calculate a variable$\hat{\alpha}$ with

$$\hat{\alpha}_t(i) = \frac{\alpha_t(i)}{\sum_{j=1}^{N} \alpha_t(j)} = C_t \alpha_t(i), \tag{2.3.42}$$

i. e. to scale each α_t by the sum of all α_t. This is done by alternatingly calculating $\bar{\alpha}_t$ and $\hat{\alpha}_t$ for the current index t, where

$$\bar{\alpha}_1(i) = \alpha_1(i), \tag{2.3.43}$$

and then inductively

$$c_t = \frac{1}{\sum_{i=1}^{N} \bar{\alpha}_t(i)}, \tag{2.3.44}$$

$$\hat{\alpha}_t(i) = c_t \bar{\alpha}_t(i), \tag{2.3.45}$$

$$\bar{\alpha}_{t+1}(j) = \sum_{i=1}^{N} \hat{\alpha}_t(i) \cdot a_{ij} \cdot b_j(O_{t+1}). \tag{2.3.46}$$

It can be seen that $\bar{\alpha}$ is initialized with the non-scaled α_1 and then each $\hat{\alpha}_t$ scales the appropriate $\bar{\alpha}_t$ by the sum over all $\bar{\alpha}_t$, which is denoted as scaling coefficient c_t. In each new step, $\bar{\alpha}_{t+1}$ must be calculated first, using the scaled $\hat{\alpha}_t$ from the previous step [24].

The difference between C_t and c_t for $t > 1$ (since $C_1 = c_1$) is that the former is calculated using the sum of all α_t while the latter uses the sum of all $\bar{\alpha}_t$. Rahimi (ibid.) also shows how C_t can be expressed in terms of c_t,

$$C_t = \frac{1}{\sum_{j=1}^{N} \alpha_t(j)}, \tag{2.3.47}$$

$$c_t = \frac{1}{\sum_{j=1}^{N} \bar{\alpha}_t(j)}, \tag{2.3.48}$$

$$C_t = \prod_{s=1}^{t} c_s. \tag{2.3.49}$$

The backward variable β is scaled by the same scaling coefficients as α (ibid.),

$$\bar{\beta_T}(i) = \beta_T(i), \tag{2.3.50}$$

$$\hat{\beta}_t(j) = c_t \bar{\beta}_t(i), \tag{2.3.51}$$

$$\bar{\beta}_t(j) = \sum_{i=1}^{N} a_{ij} \cdot b_j(O_{t+1}) \cdot \hat{\beta}_{t+1}(i). \tag{2.3.52}$$

It is also shown by Rahimi (ibid.) that the formulae for ξ and γ change to

$$\xi_t(i,j) = \hat{\alpha}_t(i) \cdot a_{ij} b_j(O_{t+1}) \cdot \hat{\beta}_{t+1}(j), \tag{2.3.53}$$

$$\gamma_t(i) = \hat{\alpha}_t(i) \cdot \hat{\beta}_t(i) \cdot \frac{1}{c_t}. \tag{2.3.54}$$

The probability of an observation sequence given the model, i. e. $P(O|\lambda)$ is no longer calculated by summing up all $\hat{\alpha}_T$ as in equation 2.3.11, since $\hat{\alpha}_T$ are scaled. $P(O|\lambda)$ could instead be calculated using the following property derived from equations 2.3.47 and 2.3.49,

$$C_T = \frac{1}{\sum_{j=1}^{N} \alpha_T(j)} = \frac{1}{P(O|\lambda)} \tag{2.3.55}$$

$$\Leftrightarrow \prod_{t=1}^{T} c_t = \frac{1}{P(O|\lambda)} \tag{2.3.56}$$

$$\Leftrightarrow P(O|\lambda) = \frac{1}{\prod_{t=1}^{T} c_t}. \tag{2.3.57}$$

However, since $P(O|\lambda)$ would be out of range, $\log(P(O|\lambda))$ is calculated instead,

$$\log(P(O|\lambda)) = -\sum_{t=1}^{T} \log(c_t). \tag{2.3.58}$$

The re-estimation formula for a_{ij} can now be written as

$$\overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i) \cdot a_{ij} b_j(O_{t+1}) \cdot \hat{\beta}_{t+1}(j)}{\sum_{t=1}^{T-1} \sum_{j=1}^{N} \hat{\alpha}_t(i) \cdot a_{ij} b_j(O_{t+1}) \cdot \hat{\beta}_{t+1}(j)}. \tag{2.3.59}$$

Re-estimation of b_j or the probability distribution function parameters is done equally by just replacing each α and β by $\hat{\alpha}$ and $\hat{\beta}$, respectively [20, 24].

Rabiner also mentions the use of logarithms for the Viterbi state sequence. Since its calculation consists of a product of probabilities, the logarithms of each of these factors can be summed up [20].

2.3.7 Multiple Observation Sequences

Rabiner [20] brings up another issue that occurs when training an HMM separately with single observation sequences when there is only a small number of observations for a state. This generally applies to recognition of handwriting, speech and sign language and needs to be solved. The re-estimation formulae are changed to comply with a set of K observation sequences, i. e. $\Theta = [O^{(1)}, O^{(2)}, \cdots, O^{(K)}]$, where each $O^{(k)} = O_1^{(k)} O_2^{(k)} \cdots O_{T_k}^{(k)}$. The goal now is to maximize

$$\begin{aligned} P(\Theta|\lambda) &= \prod_{k=1}^{K} P(O^{(k)}|\lambda) \\ &= \prod_{k=1}^{K} P_k. \end{aligned} \tag{2.3.60}$$

Van Oosten [4] shows how to build new re-estimation formulas for scaled HMMs with multiple observation sequences as input,

$$\overline{\pi}_i = \frac{\sum_{k=1}^{K} \gamma_1^{(k)}(i)}{\sum_{j=1}^{N} \sum_{k=1}^{K} \gamma_1^{(k)}(j)}, \tag{2.3.61}$$

$$\overline{a}_{ij} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^{(k)}(i) \cdot a_{ij} \cdot b_j(O_{t+1}^{(k)}) \cdot \hat{\beta}_{t+1}^{(k)}(j)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^{(k)}(i) \cdot \hat{\beta}_t^{(k)}(i) \cdot \frac{1}{c_t^{(k)}}}, \tag{2.3.62}$$

$$\overline{b}_j(l) = \frac{\sum_{k=1}^{K} \sum_{t \in [1, T_k - 1] \wedge O_t = v_l} \hat{\alpha}_t^{(k)}(i) \cdot \hat{\beta}_t^{(k)}(i) \cdot \frac{1}{c_t^{(k)}}}{\sum_{k=1}^{K} \sum_{t=1}^{T_k - 1} \hat{\alpha}_t^{(k)}(j) \cdot \hat{\beta}_t^{(k)}(i) \cdot \frac{1}{c_t^{(k)}}}. \tag{2.3.63}$$

For continuous-density HMMs, new values for μ and Σ can be calculated as follows,

$$\overline{\mu}_{jn} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \gamma_t^{(k)}(j, n) \cdot O_t^{(k)}}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \gamma_t^{(k)}(j, n)}, \tag{2.3.64}$$

$$\overline{\Sigma}_{jn} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \gamma_t^{(k)}(j, n) \cdot (O_t^{(k)} - \overline{\mu}_{jn}) \otimes (O_t^{(k)} - \overline{\mu}_{jn})^{\mathsf{T}}}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \gamma_t^{(k)}(j, n)}, \tag{2.3.65}$$

where $\gamma_t^{(k)}(j, n)$ simplifies to $\gamma_t^{(k)}(j)$ if the mixture is simple [20, 24, 4]. Van Oosten [4] explicitly uses $\overline{\mu}$ for calculating $\overline{\Sigma}$, while Rabiner does not clarify whether μ or $\overline{\mu}$ is used.

2.3.8 Initialization

Training and other basics of hidden Markov models have already been discussed. However, another very important part is initialization, i. e. determining the initial parameters of an HMM. There are several approaches that deal with this matter.

Liu et al. [10] propose three different ways of automatic initialization,

- Single random initial models,
- Multiple random initial models,
- Directly computed models.

The use of single random initial models is discouraged, as HMMs highly depend on their initial values. This is due to the Baum-Welch algorithm only being able to find a local rather than a global maximum, and thus the worse the initial choice is, the worse the trained HMM will perform.

The second approach attempts to solve this issue by generating multiple random models for initialization, distributed evenly in space. It is expected that several models near the global maximum can be found, which leads to the best possible results after performing Baum-Welch training.

As Liu et al. (ibid.) conclude, directly computing the initial model is the most convenient approach in finding the global maximum. However, they only provide formulas for HMMs with a finite number of observation symbols and do not cover continuous observation densities.

Kelly et al. [7] summarize different ways of initialization that are applicable to continuous observation densities. They describe one such approach to be manually labeling observation sequences, splitting them in a pre-defined number of states and then computing initial parameters for these states.

They refine this process by additionally automating the labeling of observation sequences, with the aim of keeping human intervention at a minimum. The goal is to find an HMM with the best combination of (S, R), where S is the number of states and R the reach of a state, i. e. the number of states that a state can transition to in an LR-model.

This is done by iterating over possible values for S and R. First, one out of K previously recorded observation sequences is chosen at random, which will be called $O^{(r)}$. A principal component analysis is performed on the sequence, then the k-means clustering algorithm is used on the principal components in order to divide the observations into clusters. Lastly, the $S - 1$ indices are found that best divide the observation sequence into S states.

For each of these states, the covariance matrix Σ and mean vector μ are calculated, then the newly created model is trained using the remaining $K-1$ observation sequences. The Viterbi algorithm helps determining the most likely state sequence of previously chosen observation sequence $O^{(r)}$, which is then used to re-align the initial S states. After this whole process, different values for S and R are chosen in order to find a pair (S, R) which provides the best possible HMM (ibid.).

2.4 Recognizing Sign Language

Automatic sign language recognition involves two major processes, namely extracting (*detecting*) features and interpreting (*recognizing*) them, as well as several problems that come along with these two. Features that can be extracted to help understanding sign language include facial expression, head posture, body posture, hand an arm location, handshape, and hand orientation [19]. However, it is not necessary to extract all these features, as good results can also be achieved by only seeing a subset of them (p. 26 in [19]).

The process of recognition usually relies on hidden Markov models [8, 6, 15, 17], while different technology has been used to detect features. This section will give an overview of previous work on both processes and then outline common problems and their solutions.

2.4.1 Overview

Early work in extracting sign language features by Takahashi and Kishino ([18]) in 1991 relied on wired gloves the user had to wear. The system was capable of recognizing handshapes, but ignored any other features and was limited to finger spelling, requiring all words to be spelt rather than signed.

Their experiments showed that 30 out of 46 pre-defined gestures of the Japanese kana manual alphabet could be recognized. For specific gestures, additional information such as "fingertips touching" were required, but not supplied by the gloves (ibid.).

A later approach in 1995 by Starner and Pentland [17] featured real-time recognition of American Sign Language from video and made use of HMMs. It required the user to wear solid colored gloves for better stability. Several restrictions were imposed, for instance some features of ASL (referencing objects by pointing at them, as well as all facial features) were ignored, and only sentences of a specific type ("personal pronoun, verb, noun, adjective, same personal pronoun") were regarded.

The system had a word accuracy of 99.5% of 395 sentences on training with grammar restrictions enabled, and 92% with grammar restrictions disabled. A problem with disabled restrictions was multiple insertion of the same sign, even if it was only signed once (ibid.).

Vogler and Metaxas ([15]) used 3D data for their system in 1998 by setting up a system of three orthogonally placed webcams, see figure 2.4.1. After recovering the body parts from video, the data were used as input for HMMs for continuous (rather than isolated) signer-dependent sign language recognition. They experimented with 2D data input and the results showed that by using 3D data, a higher word accuracy could be achieved (ibid.).

In 2007, Dreuw et al. [9] proposed a signer-independent system for sign language recognition based on speech recognition techniques, using a single webcam and thus 2D information, without the need for any gloves. After several optimazations, the word error

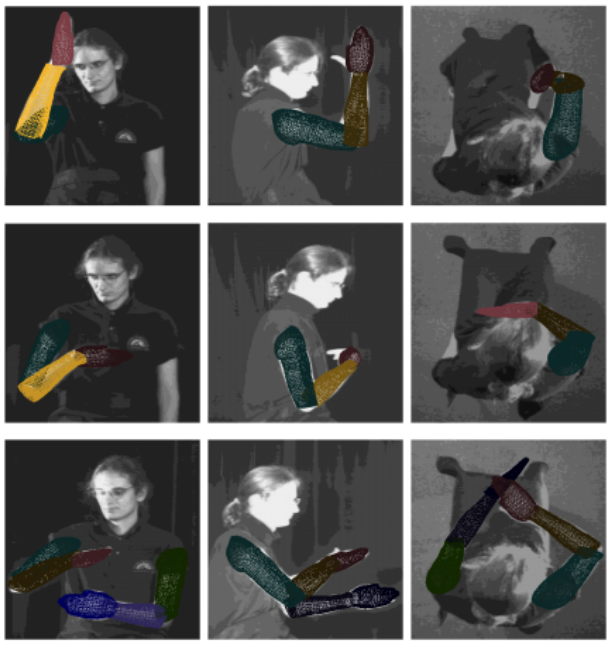

Figure 2.4.1: Vogler and Metaxas' method using three orthogonally placed webcams (figure taken from [15])

rate could be decreased to 17.9% on a vocabulary of 104 signs signed by three speakers (ibid.).

Unlike previous works that concentrated on manual features only, a more recent approach in 2009 by Kelly et al. [6] also incorporates a non-manual feature, namely head movement. The system relies on a single webcam and the user wearing colored gloves for continuous sign recognition. Testing of the framework consisted of 160 video clips of unsegmented sign language sentences and a small vocabulary of eight manual signs and three head movement gestures. A detection ratio of 95.7% could be achieved (ibid.).

The *Center for Accessible Technology in Sign* (CATS) shows a demo on how Kinect is used in CopyCat, achieving a 98.8% sentence recognition accuracy on a six-word vocabulary and awaiting any number of signs [31, 44]. CopyCat is a platform for deaf children, designed to help "develop working memory and language skills while they play the game", normally using a 2D camera in combination with wired gloves [43].

A larger ongoing project called SignSpeak is EU-funded and was introduced by Dreuw et al. [3]. It is being built on previous work, uses an ordinary 2D camera and aims at translating continuous sign language to text, supporting a large vocabulary and recognizing both manual and non-manual features [2, 3].

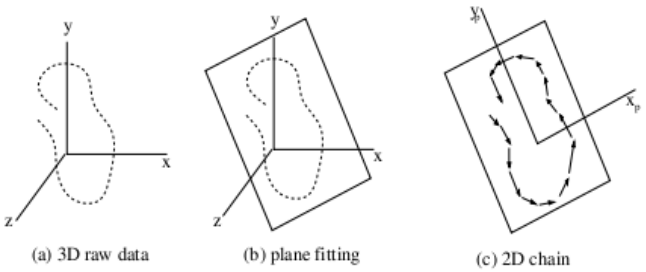

Figure 2.4.2: 3D to 2D reduction in Nam and Wohn's work (figure taken from [16])

2.4.2 Common Problems

There are several problems that need to be overcome when building a system for sign language recognition, such as

- how to extract features, i. e. how to acquire data of body parts, and combine them,
- how to handle statistical variations in the way signs are performed, either by different signers or by the same signer, and
- how to deal with movement epenthesis, i. e. the movement inbetween two signs in continuous sign language recognition.

This part will review previous work on the topic that deals with approaches on how to solve the first two issues [31, 9, 6, 17, 18, 15]. The problem of movement epenthesis recognition will not be regarded as this work concentrates on isolated sign recognition.

Extracting Features

The problem of feature extraction has already been partly discussed in section 2.4.1. Early approaches using data gloves focus on recognizing handshapes rather than movement and can thus only be used for finger spelling [18]. Since movement is one of the most important parts of sign language (p. 26 in [19]), these systems cannot be used alone for sign language recognition.

Later approaches rely on 2D cameras for feature extraction, some of requiring users to wear solid colored gloves, others detecting skin color [6, 9, 17, 15]. While most of them use a single camera and cannot identify handshapes, the solution of Vogler and Metaxas [15] has three orthogonally placed webcams acquire 3D data, resulting in more accurate recognition.

In 1996, Nam and Wohn ([16]) built a system for hand gesture recognition where 3D data of the user's hands are acquired and reduced to 2D coordinates by projecting them on a plane, as figure 2.4.2 shows.

Most work concentrates on extracting manual features only, while the system of Kelly et al. [6] also takes account of head movement using face tracking, and manages to achieve

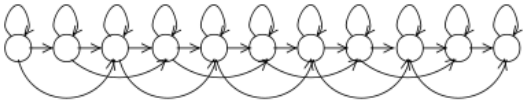

Figure 2.4.3: Optimal state model for Vogler and Metaxas' work (figure taken from [15])

a high detection ratio. Since non-manual signs are independent from manual signs, they are processed independently and the information are combined after recognition (ibid.).

The Kinect demo of CopyCat also shows high accuracy using skeletal tracking even without regarding different handshapes [31, 44].

By additionally detecting facial expressions, as planned for the ongoing SignSpeak project [2], some important grammatical features and non-manual adverbs (pp. 99-104 in [19]) can be taken into account.

Handling Statistical Variations

Even if signed by the same person, a sign is hardly ever performed twice in exactly the same way. These statistical variations, both intra- and inter-personal, need to be handled for convenient sign recognition. This is usually done by introducing HMMs, as previous work shows [6, 15, 16, 17].

Due to their state-based character, changes of a sign over time as well as varying durations of the same sign can be described. Vogler and Metaxas determined the optimal state model for their work to be an LR-model with reach 2, as shown in figure 2.4.3 [15].

Output probabilities are determined by a logarithmically concave function in continuous density HMMs, or by a weighted mixture of such. Kelly et al. [6] use an M-dimensional multivariate normal distribution for M-dimensional observation vectors $O_t = \{o_1, o_2, \cdots, o_M\}$,

$$\mathcal{N}(O_t; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{M}{2}} \cdot |\Sigma|^{\frac{1}{2}}} \cdot \exp(-\frac{1}{2}(O_t - \mu)^{\mathsf{T}} \cdot \Sigma^{-1}(O_t - \mu)), \tag{2.4.1}$$

where O_t is part of an observation sequence $O = O_1 O_2 \cdots O_T$, μ represents the mean vector and Σ the covariance matrix. The elements $o_1, \cdots, o_M$ of the feature vector O_t may represent data from feature extraction, such as "right hand position" or "left shoulder position" (ibid.).

Kelly et al. also use different HMMs for each hand, and another independent HMM for head gestures. They propose the 5-dimensional feature vector to consist of the hand position relative to the eyes (RP_x, RP_y), the direction of hand movement (V_x, V_y), and the distance between both hands (D_H), resulting in a feature vector $O_t = (RP_x, RP_y, V_x, V_y, D_H)$ (ibid.). Training an HMM for a specific sign is done as described in section 2.3.8.

3 Kinect-based Sign Language Recognition

The goal of this work is to implement a framework for recognition of isolated signs. More generally, this framework will be able to recognize any isolated gestures, but will be tested with signs of German Sign Language (DGS, *Deutsche Gebärdensprache*).

One difference as compared to gestures is the similarity among several signs. While gesture recognition usually deals with artificially created gestures to control devices or software, these gestures can vary greatly among each other, and thus can be distinguished more easily. Moreover, gestures are generally less complex than signs and do not involve as many different handshapes, nor do they include movement of lips or the head.

These features will not be recognized in this work either. However, in order to allow for high recognition rates, experiments with other features will be made that are offered by Kinect and are uncommon in recognition systems with 2D cameras.

3.1 Project Overview

The framework created as part of this work is called Dragonfly – **Dra**w **g**estures **on** the **fly** – and mainly consists of two classes that users can integrate in their software. These classes are *DepthCamera* and Dragonfly. The former serves as an interface to OpenNI, i. e. it updates the camera image and supplies data of the skeleton joints – the body parts –, while the latter processes these data for gesture recognition. The major part of the framework is shown in the class diagram in figure 3.1.1.

For gesture recognition, the framework makes use of continuous density HMMs, which were implemented as part of this thesis as well. The implementation offers features such as automatic initialization, Baum-Welch re-estimation with multiple observation sequences, and serialization.

ObservationSeqProbs are associated with both HMMs and observation sequences. The name is an abbreviation for "observation sequence probabilities", as they link the data between an observation sequence and an HMM by calculating the (logarithmical) probability of the sequence given the model. They are also capable of computing the Viterbi state sequence.

In addition, the framework includes an own implementation of the k-means clustering algorithm. However, since it only accounts for a small part, its classes are not listed in

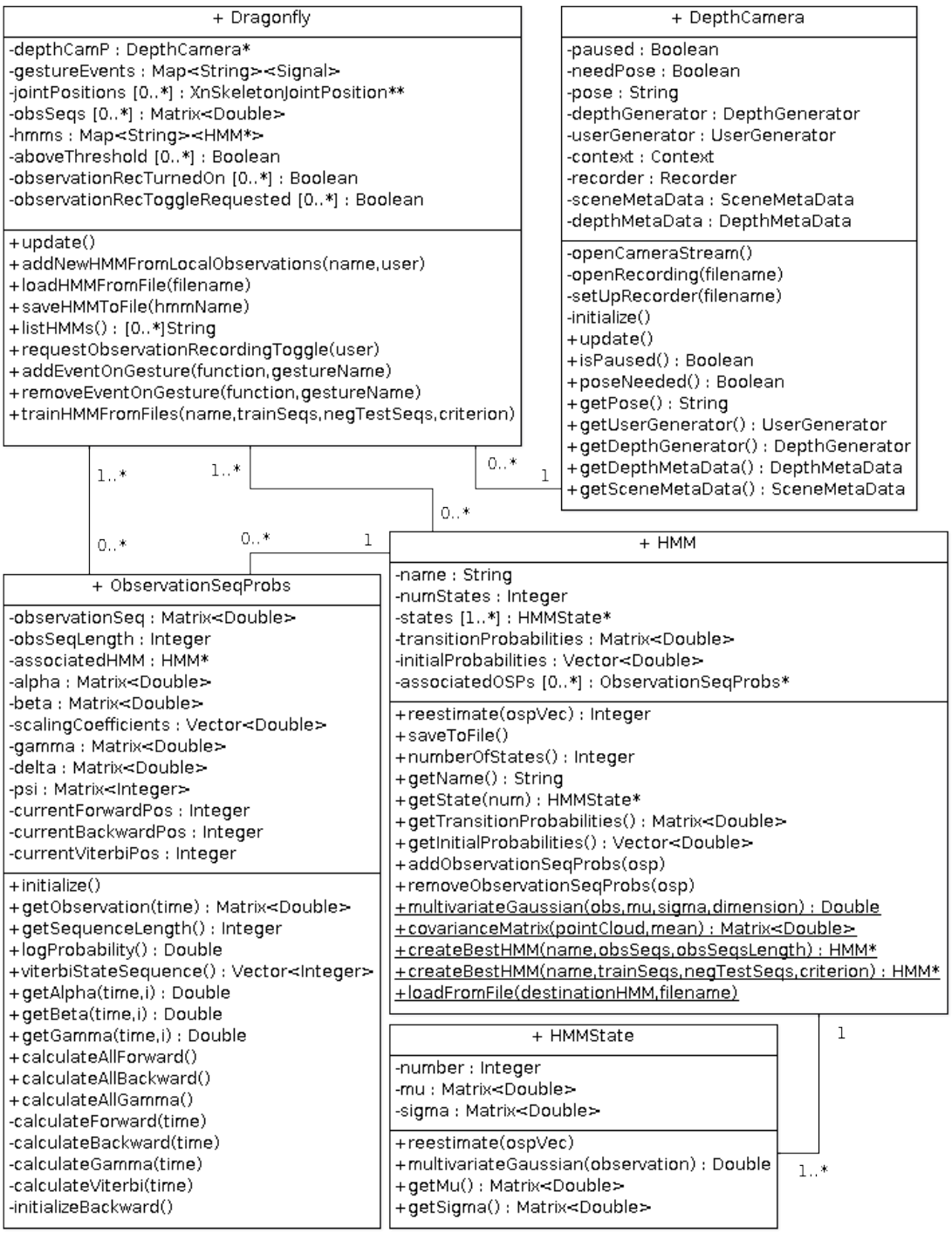

Figure 3.1.1: UML class diagram of Dragonfly

the diagram nor will they be explained in detail. The functionality is described in [5] and various other sources.

The major parts of the framework is descibed more detailedly throughout this chapter.

3.2 Implementation of Hidden Markov Models

The HMM implementation comprises probabilities for observation sequences, HMMs themselves, and states of HMMs. An HMM can be created either completely manually – by defining every single parameter by hand –, partly manually by giving an observation sequence and split indices, or automatically with just observation sequences as input. The number of states and the name have to be defined for any of these methods. Alternatively, HMMs can be loaded from a file using the static `loadFromFile` function.

Each state of an HMM is represented by an *HMMState* object that contains its state number. Since only continuous density HMMs are regarded, they feature a mean vector μ and a covariance matrix Σ, as explained in 2.3.5. The logarithmically concave probability density function used is a simple-mixture (i. e. $M = 1$) multivariate Gaussian function[1]:

$$\mathcal{N}(O_t; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{M}{2}} \cdot |\Sigma|^{\frac{1}{2}}} \cdot \exp(-\frac{1}{2}(O_t - \mu)^{\mathsf{T}} \cdot \Sigma^{-1}(O_t - \mu)).$$

Partly manual creation works as follows:

- Provide a name, the number of states N, and one observation sequence. Also provide $N - 1$ split indices that show at which points in time state changes occur.
- Set initial probabilities π – since an LR-type HMM is used,

$$\begin{aligned} \pi_0 &= 1 \\ \pi_i &= 0 \qquad , i > 0. \end{aligned}$$

- Calculate transition probabilities A. When $N-1$ split indices are input, an LR-type HMM with N states and reach $R = 1$ will be created, where

$$\begin{aligned} a_{ii} &= \frac{\#\text{observations in state } S_i - 1}{\#\text{observations in state } S_i} && , 0 \leqslant i < N - 1 \\ a_{i(i+1)} &= 1 - a_{ii} && , 0 \leqslant i < N - 1 \\ a_{(N-1)(N-1)} &= 1 \\ a_{ij} &= 0 && , 0 \leqslant j < i \vee j > i + 1. \end{aligned}$$

- Set up S point clouds using the split indices. The mean vector μ and covariance matrix Σ of each such point cloud are calculated and set up for the approprate state.

[1] The *Vision Numerics Libraries* are used for vector operations, available at *http://vxl.sourceforge.net/*

Automatic creation is similar, however split indices are no longer provided. Some of the steps proposed by Kelly et al. [7] (described in section 2.3.8) are used, however they did not provide a way of determining split indices. Thus, an ergodic model is used with this initialization method.

- Provide a name, the number of states N, and a set of observation sequences. Other than the method proposed by Kelly et al., this approach uses all given observation sequences instead of one randomly chosen sequence.
- Create a point cloud with every single observation from every sequence as a point.
- Perform k-means clustering on the point cloud, initial prototypes are chosen randomly among its points. Empty clusters are avoided by determining a new prototype from the biggest cluster. The point chosen is the one farthest away from that cluster's prototype.
- Sort the resulting clusters – each cluster represents a state. The first observation sequence of the set is used to determine what clusters its points belong to. Whichever cluster is encountered next that has not been encountered before, is chosen as the next state.
- Calculate the transition probability distribution A using data from all observation sequences, where

$$a_{ij} = \frac{\#\text{transitions from } S_i \text{ to } S_j}{\#\text{transitions from } S_i} \qquad , 0 \leqslant i, j < N.$$

- Compute the initial probability distribution π, where

$$\pi_i = \frac{\#\text{observation sequences starting in } S_i}{\#\text{observation sequences}} \qquad , 0 \leqslant i < N.$$

- Set each state's μ to the mean vector of the corresponding cluster.
- Determine Σ for each state by calculating the covariance matrix for each corresponding cluster.

After initializing an HMM, an *ObservationSeqProbs* object ("sequence probabilities" or "probabilities") can be created for each observation sequence.

This object links an observation sequence to an HMM and wraps calculation of several probabilities around it, such as the scaled forward and backward variables, and thus also the logarithmic probability of the sequence given the HMM. By creating sequence probabilities for several HMMs using the same observation sequence, it can be determined which HMM best matches the given sequence.

The Viterbi state sequence can also be calculated by probabilities objects. This is done using logarithms to prevent underflows, since the terms can be very close to 0 (see section 2.3.6).

With a set of sequence probabilities, an HMM can be re-estimated using the Baum-Welch algorithm, as explained in sections 2.3.3, 2.3.5 and 2.3.7.

Additional implementation issues were encountered during re-estimation, when some sequence probabilities could not be calculated due to underflow, although scaled HMMs are being used. Since generally only a small amount of sequences is critical and cannot be used for re-estimation, these sequences are filtered. The `reestimate` method returns the number of these critical sequences. That way, functions calling `reestimate` know whether it was successful, and can work around the issue in case the amount of critical sequences is too high.

The HMM implementation offers a static function called `createBestHMM` that will be explained in section 3.4.

3.3 Framework Implementation

To create an instance of the framework, a *DepthCamera* object is needed. This is the actual interface to OpenNI and could easily be replaced by a similar implementation that uses a different library for skeleton data. A DepthCamera can either display the camera's current depth stream, or play a previously recorded file which is automatically looped. In addition, the depth stream can also be recorded to a file. The `update` method should be called in a loop as it updates the camera's image.

A DepthCamera supplies the framework with data of the users' joint positions – these are three-dimensional vectors containing the coordinates of each body part, e. g. *left hand*, *right elbow*, or *head*. They can be processed and then serve as observations.

The framework itself also offers an `update`-method which needs to be called at each frame. It stores the current joint positions as well as those of the previous three updates for each user. The `update`-method is descibed in figure 3.3.1, however for simplification, this graphic only shows the process for one user.

Observations are recorded[2] for every user separately, and consist of an N-dimensional feature vector. This can be data such as *velocity* or *absolute position* of each hand, or *distance between hands*, among others.

Determining whether observations should be recorded or not works by defining a threshold for the strong hand (e. g. the right hand). Whenever that hand is above the given threshold, for example the torso's Y-position, observations are recorded at each frame and stored in a local matrix. When the hand is below the threshold, recording is stopped and the matrix is no longer updated with observations. Several of these matrices are stored in a list called `obsSeqs`, an abbreviation for "observation sequences".

`requestObservationRecordingToggle` can be called if the user does not wish to have observations recorded. This method immediately toggles observation recording as

[2] Recording observations does not mean recording the camera's depth stream, both can be done independently.

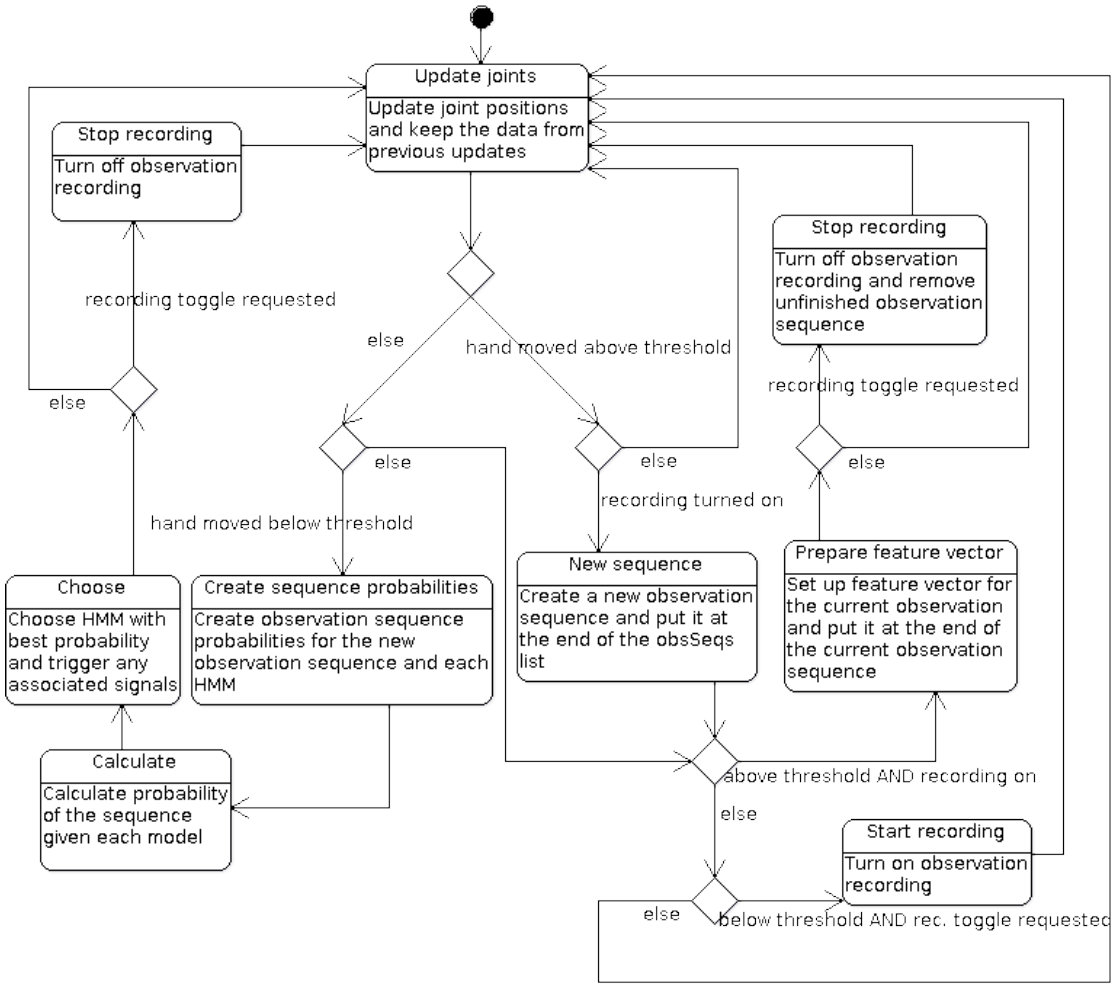

Figure 3.3.1: Statechart diagram of the `update`-method (for one user)

soon as the user is below the defined threshold, and automatically deletes the unfinished observation sequence in case it was called while recording.

If HMMs have already been added to the framework, every time the user's hand moves below the threshold, sequence probabilities of the newly recorded sequence are calculated for each existing HMM. The probability of the same sequence, given each model, is calculated, and the results are compared. This allows determining the model that best matches the sequence.

To make use of this information, a system for callback functions has been implemented. Since HMMs must have distinct names, these can be uniquely associated with a signal[3] using a hashmap. A signal can be linked to or unlinked from an HMM by calling `addEventOnGesture` or `removeEventOnGesture`, respectively, providing both the HMM's name and a pointer to the callback function.

HMMs support serialization and can easily be saved to a file using `saveHMMToFile`. The appropriate method to load an HMM is called `loadHMMFromFile`.

For easier testing, an extended feature vector sequence has been implemented, which is not shown in the above figures. These extended vector sequences work exactly as the ordinary observation sequences, and are stored in a list similar to `obsSeqs`. They are also updated at each frame, however a much larger number of features can be stored.

The list of these sequences can be saved at any time, and loading it extracts a smaller amount of features, from which ordinary observation sequences can be generated. This way, testing can be done much faster, since any OpenNI recording of a depth stream can be played once and HMMs can be created with several different feature vectors by extracting different parts of the extended vector.

3.4 Initialization and Training

Creating new gestures and training them is done successively in one of several provided ways. They rely on a function called `createBestHMM` that is similar to parts of the method by Kelly et al. as described in 2.3.8, but also modifies and extends it.

HMMs are created by providing a maximum number of states, a set of observation sequences – used for initialization and re-estimating – and a set of negative test sequences that do not contain the actual gesture to be trained. An example of one out of several similar training sequences is shown in figure 3.4.1.

Cross validation splits observation sequences into sequences actually used for training (two third), and positive test sequences the model should recognize correctly without having them used for training (one third). Additionally, a set of existing HMMs that represent different gestures can be provided.

[3] The boost C++ libraries provide signals as an implementation of the observer pattern.

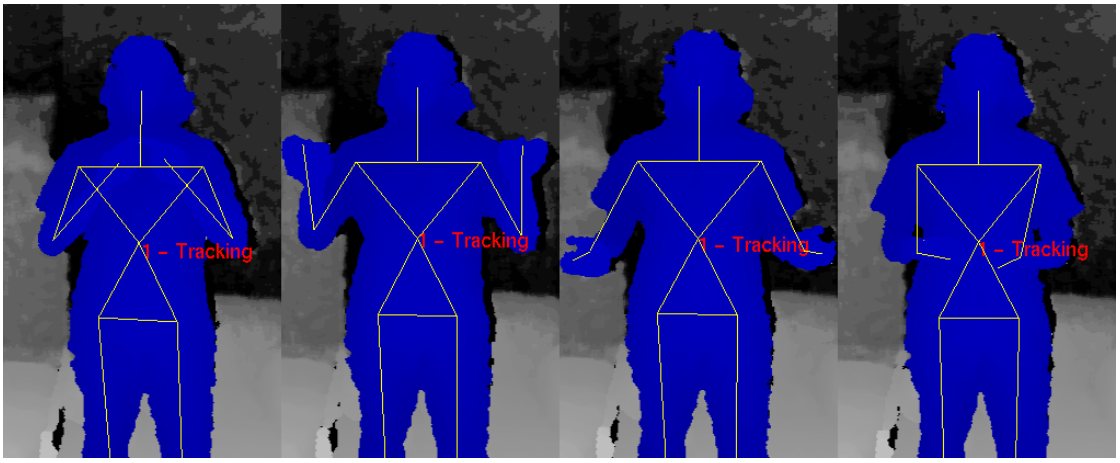

Figure 3.4.1: Example training sequence of the sign PAKET (packet)

The algorithm then initializes and trains several HMMs using the training sequences, and determines the optimal HMM given the rest of the data, according to an optimality criterion that must be defined as well. In detail, the algorithm works as follows:

1. Split provided observation sequences into training and positive test sequences according to one of the ways shown in figure 3.4.2. Two third are used for training and one third for positive testing.
2. Initialize with $S = 1$ states. Set the best HMM to `NULL` and the best value for each optimality criterion to the worst possible value.
3. Create $M = 5$ HMMs, each with S states, from the given set of training sequences using automatic initialization. During this initialization, k-means clustering is performed which may deliver different results each time due to random initial prototypes. Hence, the creation process is repeated several times.
4. Re-estimate these HMMs by the Baum-Welch algorithm, using the same training sequences as input.
5. Set $m = 1$.
6. Among the M created HMMs, choose the one at position m.
7. If at least half of the observation sequences could not be processed due to underflow, discard this HMM and go to step 10.
8. Determine the values for all optimality criteria given the newly created model and all provided data, such as positive and negative test sequences, and HMMs of other gestures.
9. Update the best value for each optimality criterion. If this HMM is better than the stored best HMM according to the chosen criterion, define it as the new best HMM.

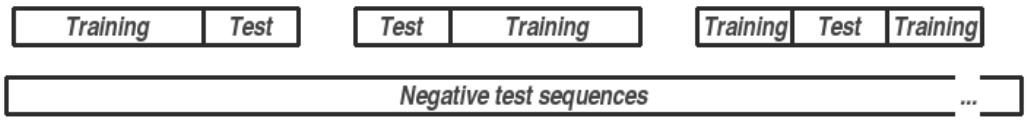

Figure 3.4.2: Sequences used as input for HMM initialization

10. Increment m. If $m \leqslant M$, go to step 6.
11. Increment S. If $S \leqslant N$ (where N is the maximum number of states), go to step 3.
12. If any the three split combinations is left, split observation sequences accordingly and go to step 2.
13. Return the best HMM (which is `NULL` in case the procedure failed to create any HMM at all).

This procedure guarantees to deliver a re-estimated HMM that best matches the given data for the chosen criterion, depending on how well k-means clustering performs. Possible optimality criteria are *best recognition rate*, *highest variance*, and *lowest negative above worst positive rate*.

The first only uses positive test sequences and tests them with the newly created HMM and all other HMMs. Negative test sequences are neglected. Each positive test sequence is tested with all HMMs and the number of correct results is saved and then summed up. A test result is correct when the sequence given the new HMM has a higher probability than the sequence given any other HMM. The summed number is divided by the total number of tests, and the resulting *recognition rate* is to be maximized.

The second criterion calculates the average logarithmic probability of all negative test sequences given the new model, and subtracts it from the average logarithmic probability of all positive test sequences given that model. Gestures of the new model can be distinguished from other gesture more clearly the higher this value is.

Determining the *lowest negative above worst positive rate* is done by saving the lowest probability of any positive test sequence given the new model – i. e. of the worst positive test sequence. Then, probabilities of all negative test sequences are calculated for the model. The number of negative test sequences, with a probability higher than that of the worst positive test sequence, is divided by the total number of negative test sequences. The lower this value gets, the better the success rate of the new HMM is.

For equal values of the third criterion, the second criterion is used to determine which HMM is better. HMMs should be created by saving the extended feature vector and using the `trainHMMFromFiles` method.

However, the list of locally saved observation sequences can also be used to create a new HMM by calling `addNewHMMFromLocalObservations` from outside the `update`-method. This method processes the data inside `obsSeqs` for use with `createBestHMM`.

Since no negative sequences can be provided this way, the *best recognition rate* criterion is used.

3.5 Using the Framework

The framework is meant to be integrated into other software that uses its supplied methods accordingly. However, a small application for testing has been created as part of this work in order to demonstrate the usage.

The application incorporates a part of OpenNI that displays the camera's depth stream in an OpenGL window using GLUT[4].

Other features include the ability to set up a depth camera for displaying as well as recording its stream, or playing a previously recorded file, by either passing `--record` or `--play`. The framework is updated at 60 frames per second, and the keyboard can be used for additional controls.

Pressing `p` pauses or resumes the stream, the numbers `1` to `4` toggle observation recording for the appropriate user. Recorded observations can be used to create an HMM by pressing `t`, and HMMs can be loaded or saved by pressing `l` or `s`, respectively. The above mentioned extended feature vector sequences can be saved by pressing `o` and loaded by pressing `i`. Extracting a smaller set of features, however, is currently hard-coded, as using `trainHMMFromFiles` is.

Whenever needed, the application will ask for input on the console. Pressing `Esc` terminates the program. An example of the application and its output, after performing the sign DANKE (thank you) once and GIRAFFE twice, can be seen in figure 3.5.1.

The source code of this framework as well as the testing application can be found on the appended disc.

[4]The *OpenGL Utility Toolkit*, available at *http://www.opengl.org/resources/libraries/glut/*.

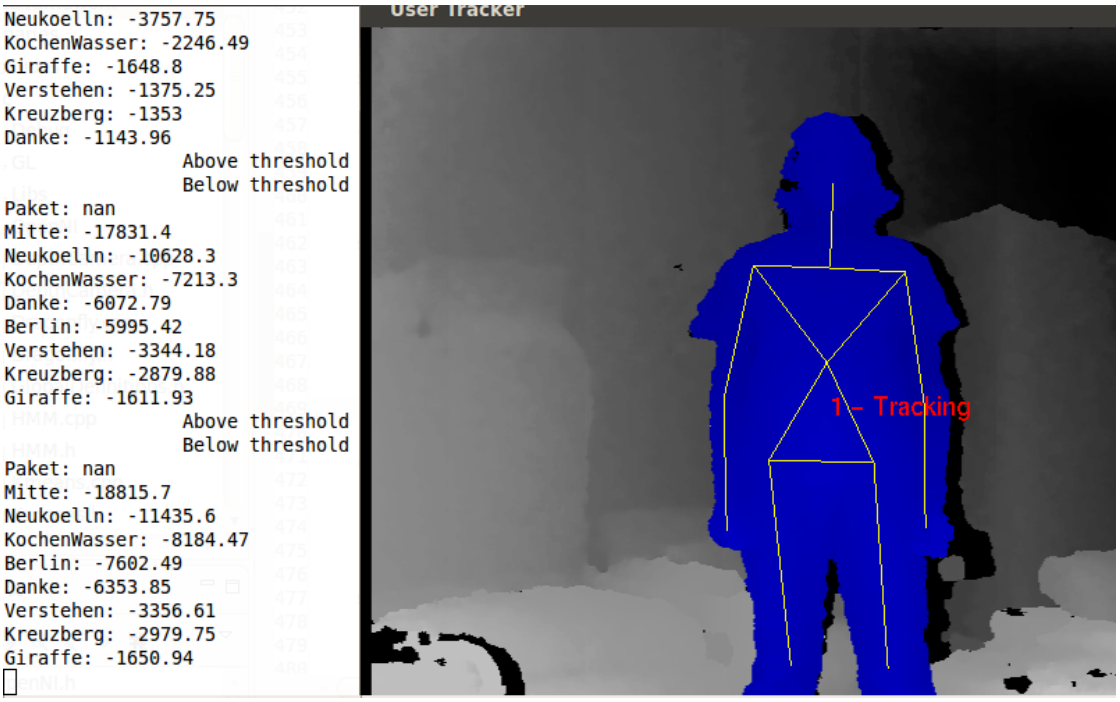

Figure 3.5.1: Example output after performing three signs

4 Experiments and Evaluation

With a testing application ready, sign language experiments have been started. Using the extended feature vector, several features have been saved. These are:

- 2-dimensional positions of both hands relative to the neck (x_{abs}, y_{abs}),
- The distance between both hands,
- 2-dimensional movement of both hands (x_{rel}, y_{rel}), i. e. their position relative to the position two updates before,
- 2-dimensional positions of both elbows relative to the neck,
- The absolute velocity of each hand, i. e. $v = \sqrt{x_{rel}^2 + y_{rel}^2}$,
- The absolute distance of each hand to the neck, i. e. $d = \sqrt{x_{abs}^2 + y_{abs}^2}$, and
- 2-dimensional normalized velocity of each hand, which is calculated as follows:

$$(v_x, v_y) = \begin{cases} (0,0) & , x_{rel} = 0 \wedge y_{rel} = 0 \\ \left(\frac{x_{rel}}{|x_{rel}| + |y_{rel}|}, \frac{y_{rel}}{|x_{rel}| + |y_{rel}|} \right) & , \text{else} \end{cases} .$$

Isolated experiments have been made to determine the most suitable feature vector, which is seven-dimensional and consists of (x_{abs}, y_{abs}) for both hands, v_{right}, v_{left}, and the x-position of the right elbow. The elbow's position helps distinguish signs where the hand is near the head, but differently rotated, such as in the sign BERLIN as compared to VERSTEHEN (to understand). The elbow is then either close to the body or spread out.

With the determined feature vector, a selection of nine signs has been performed by a signer of DGS, and for each sign an HMM has been created. These signs are tested with separate recordings of the same signer. Another test is done by a non-fluent signer who has been taught the signs. It shows how the HMMs perform when used by a person which has not trained them. In a second run, the same nine signs are trained with sequences of both signers and then tested by them.

The results of these three tests will be evaluated. In both runs, HMMs are initialized automatically with up to ten signs. The signs used are BERLIN, DANKE (thank you), GIRAFFE, KOCHEN (to boil), KREUZBERG, MITTE, NEUKÖLLN (districts in Berlin), PAKET (packet), and VERSTEHEN (to understand). Six of these signs are

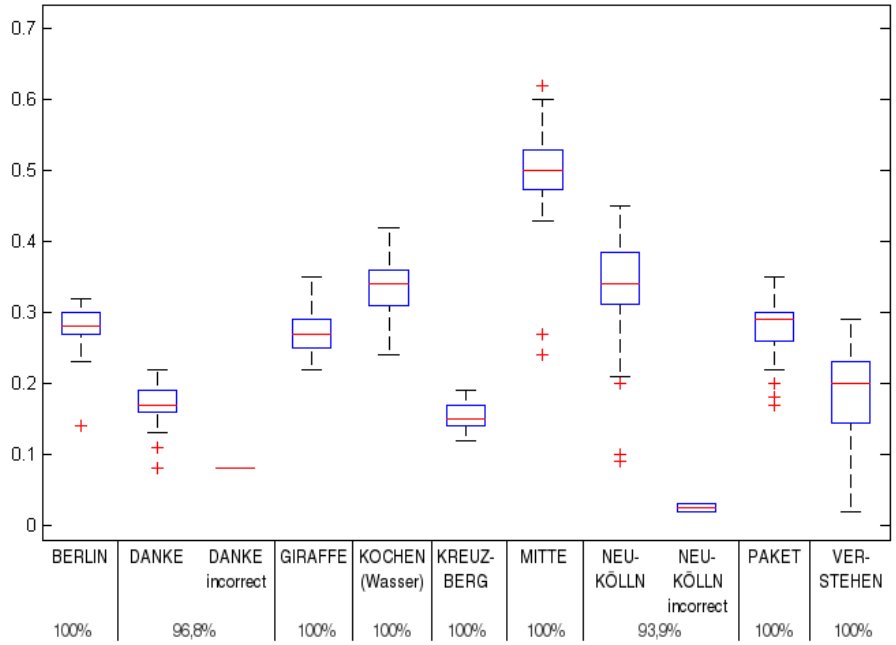

Figure 4.0.1: Results of the DGS signer testing the previously trained signs

performed with one hand (both persons' right hands), the other three are performed with two hands. Information on how to perform these signs is included in the appendix.

The results have been normalized to a value between 0 and 1 by artificially defining the best possible probability as $b = -700$ and setting it to 1, and then normalizing all other values x by

$$x \mapsto \frac{b}{x}.$$

Each test consists of 30-45 test sequences. To keep the data simple, it is reduced to the important parts. All figures show the recognition rate of each sign at the bottom, and a box plot.

If a sign was recognized correctly, the box plot shows the distance between the (normalized) probability of the recognized sign and the probability of the second best sign. In case a sign was not recognized correctly (indicated by the label '*incorrect*'), the distance between the recognized sign's probability and the actually performed sign's probability is shown.

Figure 4.0.1 shows results of the first run with tests of the same signer of DGS who has trained the signs beforehand.

Except for DANKE and NEUKÖLLN, all signs are recognized with a rate of 100%. The three two-handed signs (KOCHEN(Wasser), MITTE and PAKET) show a higher

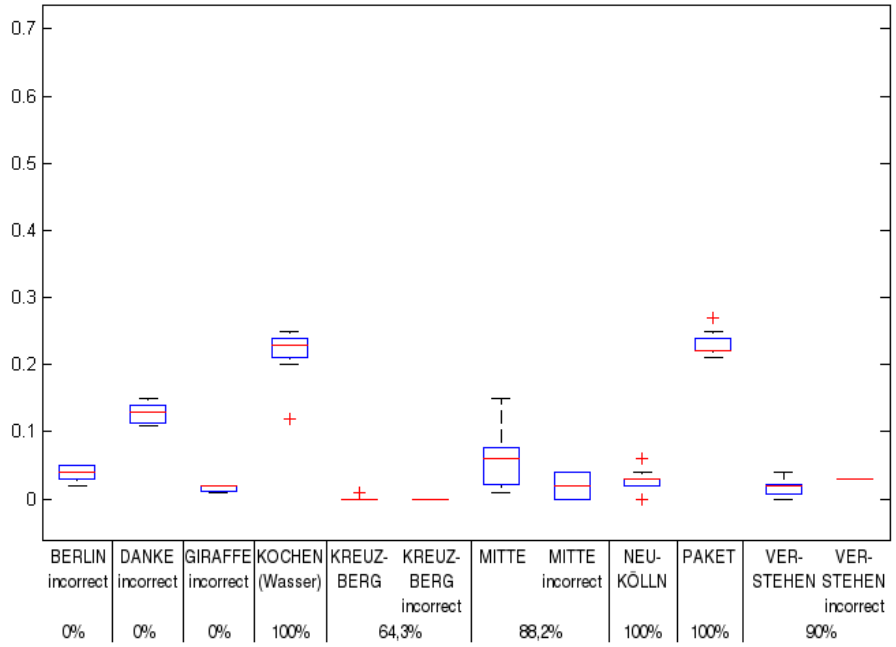

Figure 4.0.2: Results of the non-fluent signer testing the DGS signer's trained signs

distance to the second best sign than the rest, except NEUKÖLLN which at the same time features the worst recognition rate of 93,9%. Incorrect recognitions are usually close to the actually performed signs' probabilities.

Results of the non-fluent signer testing the DGS signer's trained signs are shown in figure 4.0.2.

It can be seen that the results are significantly worse. The first three signs could not be recognized at all, while the two-handed signs, as well as NEUKÖLLN and VERSTEHEN show a rather high recognition rate. Recognizing KREUZBERG rather than something else is very close. Distances to the second best signs are generally lower than in the first test.

The final test run uses a different set of HMMs, trained not only by the signer of DGS, but also by the non-fluent signer. Results are shown in figure 4.0.3.

The recognition rates are much better than in the second test, although the distances to the second best signs are generally lower than in the first test. The signs offering the highest distances – KOCHEN(Wasser), MITTE, NEUKÖLLN and PAKET – are the same as in the first test.

The only exception is the HMM for DANKE which features the worst performance – all of its tests have a logarithmic probability of $-\infty$, and thus ∞ distance to the second best sign. This may be due to the way DANKE is performed, which mainly involves z-movement of the strong hand as compared to the other signs.

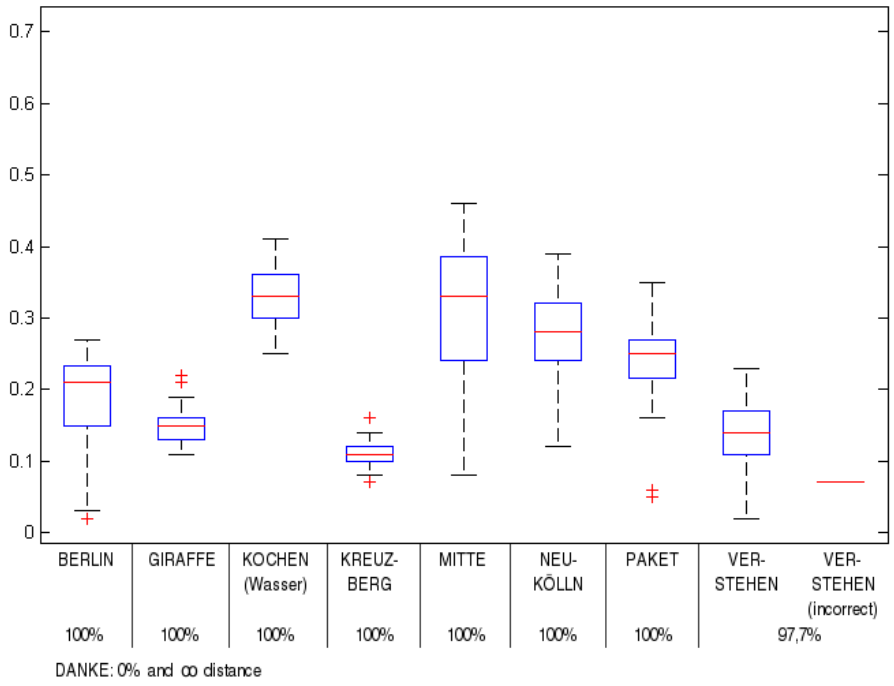

Figure 4.0.3: Results of the third test run with signs trained by both signers

The overall results show that using only a single person for training, the trained HMMs are very signer-dependent. This can be avoided by training HMMs with observation sequences of several people performing signs differently. However, the recognition rates are not yet optimal and could be improved by testing yet more feature vectors and having more than two signers train each HMM, but also by improving other parts of the software.

5 Conclusion and Future Work

This work has shown that Kinect and depth cameras in general are well-suited for sign language recognition. They offer 3D data of the environment without a complicated camera setup and efficiently extract the users' body parts, allowing for recognition of not just hands and head, but also other parts such as elbows that can further help distinguish similar signs.

Another advantage is the independency of lighting conditions, as the camera uses infrared light. All body parts are detected equally well in a dark environment and there is no need for the user to wear special-colored gloves or wired gloves.

As part of this work, a framework for isolated gesture recognition has been created, and tested with signs of German Sign Language. It features hidden Markov models with continuous observation density, and initialization methods for HMMs that deliver the best suited model from k-means clustering results, given an optimality criterion as well as positive and negative testing data. The models need to be scaled in order to prevent underflow, and are trained with multiple observation sequences using the Baum-Welch algorithm.

Experiments have shown that when training signs with only one signer, the resulting HMMs are very signer-dependent, and five out of nine signs are only recognized reliably by the person who has trained them.

In a subsequent experiment, the same nine signs have been initialized and trained using observation sequences of two people. The results show a more convenient recognition rate, with eight of nine signs being recognized at least at 97,7%. The remaining sign could not be recognized at all, which is likely due to the fact it involves mainly z-movement rather than x- and y-movement.

Accurately recognizing sign language, however, not only involve tracking hands. There are signs that only differ in facial expression or handshape and are similar otherwise. Facial expression, body posture and head movement are often used to express grammatical features that are essential for continuous sign language recognition.

A technical demo and an announced commercial framework are supposed to feature finger detection and may improve future sign language recognition. Facial expressions can be recognized using Kinect's integrated RGB camera, however this would require a well-lit environment.

Finally, the next step is to enhance the developed framework to feature continuous sign language recognition. This involves detecting a sign's start and end position, as well as

movement epenthesis as described by Kelly et al. [7] in order to distinguish between hand movement within a sign and between two signs.

In conclusion, this work shows that the approach of sign language recognition with depth cameras is worth further consideration and features its own advantages, while leaving room for improvement of both the developed framework itself as well as underlying technology.

Parts of these results have been submitted at the International Conference on Artificial Intelligence and Soft Computing 2012 [1].

Appendix

Information on how to perform the signs trained in the framework is provided below. The signs BERLIN, PAKET and VERSTEHEN are only available as videos which can be found on the included disc. The sign DANKE can be found at [26].

NEUKÖLLN is performed by putting the hand near the heart and moving it in a very small circle. KREUZBERG is composed of the signs KREUZ and BERG. The former can be found on the included disc. The remaining signs are shown in figures 0.1 to 0.4.

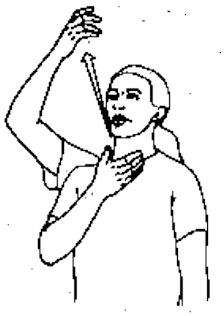

Figure 0.1: The sign GIRAFFE in DGS (figure taken from [25])

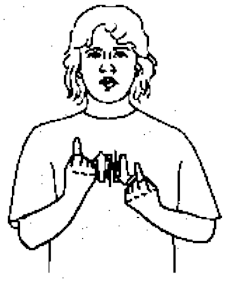

Figure 0.2: The sign KOCHEN(Wasser) in DGS (figure taken from [25])

Figure 0.3: The sign BERG in DGS (figure taken from [25])

Figure 0.4: The sign MITTE in DGS (figure taken from [25])

Bibliography

[1] Lang S., Block-Berlitz M., Rojas R.: "*Sign Language Recognition with Kinect*", IN: International Conference on Artificial Intelligence and Soft Computing (ICAISC), 2012 (to appear)

[2] Martínez Ruíz G. (CRIC): "*Scientific understanding and vision-based technological development for continuous sign language recognition and translation, System Specifications and the Corpus Needs (v2)*", SignSpeak Minor Public Deliverable, 2010

[3] Dreuw P., Forster J., Gweth Y., Stein D., Ney H., Martínez G., Vergés-Llahí J., Crasborn O., Ormel E., Du W., Hoyoux T., Piater J.,Moya J. M., Wheatley M.: "*SignSpeak – Understanding, Recognition, and Translation of Sign Languages*", IN: 4th Workshop on the Representation and Processing of Sign Languages: Corpora and Sign Language Technologies, Language Resources and Evaluation Conference (LREC), 2010

[4] van Oosten J.-P.: "*Can Markov properties be learned by hidden Markov modelling algorithms?*", Master Thesis supervised by Prof. Dr. Lambert Schomaker and Dr. Marco Wiering, Department of Artificial Intelligence at University of Groningen, The Netherlands, 2010

[5] Block-Berlitz M.: "Java Intensivkurs – In 14 Tagen lernen Projekte erfolgreich zu realisieren", ISBN: 978-3-64203-954-6, 2nd edition, Springer-Verlag Berlin Heidelberg, 2009

[6] Kelly D., Delannoy J. R., McDonald J., Markham C.: "*A Framework for Continuous Multimodal Sign Language Recognition*", IN: Proceedings of the 2009 International Conference on Multimodal Interfaces, ISBN: 978-1-60558-772-1, ACM, pp. 351-358, 2009

[7] Kelly D., McDonald J., Markham C.: "*Recognizing Spatiotemporal Gestures and Movement Epenthesis in Sign Language*", IN: 13th International Machine Vision and Image Processing Conference (IMVIP '09), ISBN: 978-0-76953-769-2, IEEE Computer Society Washington, DC, USA, 2009

[8] Cheng C., Sha F., Saul L. K.: "*Matrix Updates for Perceptron Training of Continuous Density Hidden Markov Models*", IN: Proceedings of Twenty Sixth International Conference on Machine Learning (ICML-09), ISBN: 978-1-60558-516-1, ACM, pp. 153-160, 2009

[9] Dreuw P., Rybach D., Deselaers T., Zahedi M., Ney H.: "*Speech Recognition Techniques for a Sign Language Recognition System*", IN: INTERSPEECH 2007, 8th

Annual Conference of the International Speech Communication Association (ISCA 2007), pp. 2513-2516, 2007

[10] Liu N., Davis R. I. A., Lovell B. C., Kootsookos P. J.: "*Effect of Initial HMM Choices in Multiple Sequence Training for Gesture Recognition*", IN: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC '04), ISBN: 978-0-76952-108-4, Vol. 2, IEEE Computer Society, pp. 608-613, 2004

[11] Dittrich G., Kiyomi T., Tanaka Y.: "*Minna no nihongo I: Übersetzungen & grammatikalische Erklärungen*", ISBN: 978-4-88319-239-7, 3A Corporation, 2002

[12] *Gesetz zur Gleichstellung behinderter Menschen (Behindertengleichstellungsgesetz - BGG)*, Bundesministerium der Justiz, Ausfertigungsdatum: 27.04.2002

[13] Li X., Parizeau M., Plamondon R., "Training Hidden Markov Models with Multiple Observations – A combinatorial Method", IN: IEEE Transactions on PAMI, Vol. PAMI-22, No. 4, pp. 371-377, 2000

[14] Starner T., Weaver J., Pentland A.: "*Real-Time American Sign Language Recognition Using Desk and Wearable Computer Based Video*", IN: IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 12, pp. 1371-1375, 1998

[15] Vogler C., Metaxas D.: "*ASL Recognition Based on a Coupling Between HMMs and 3D Motion Analysis*", IN: Proceedings of the Sixth International Conference on Computer Vision, ISBN: 978-8-17319-221-0, Narosa Publishing House, pp. 363-369, 1998

[16] Nam Y., Wohn K.:"*Recognition of Space-Time Hand-Gestures using Hidden Markov Model*", IN: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, pp. 51-58, 1996

[17] Starner T., Pentland A.: "*Real-Time American Sign Language Recognition from Video Using Hidden Markov Models*", IN: ISCV '95 Proceedings of the International Symposium on Computer Vision, ISBN: 978-0-81867-190-6, IEEE Publications, U. S., pp. 265-270, 1995

[18] Takahashi T., Kishino F.: "*Hand Gesture Coding Based On Experiments Using A Hand Gesture Interface Device*", IN: ACM SIGCHI Bulletin, Vol. 23, No. 2, pp. 67-73, 1991

[19] Boyes Braem P.: "*Einführung in die Gebärdensprache und ihre Erforschung*", ISBN: 978-3-92773-110-3, 1st edition, SIGNUM-Verlag, IN: Internationale Arbeiten zur Gebärdensprache und Kommunikation Gehörloser, No. 11, 1990

[20] Rabiner L. R.: *"A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition"*, IN: Proceedings of the IEEE, Vol. 77, No. 2, pp. 257-286, 1989

[21] Wilbur R. B.: "*American Sign Language and Sign Systems*", ISBN: 978-0-83910-994-5, 1st edition, University Park Press, 1979

[22] Battison R.: "*Lexical Borrowing in American Sign Language*", ISBN: 978-0-93213-002-0, Linstok Press, 1978

[23] Szczepanski M.: "GebLex – ein Gebärdenlexikon", dictionary of German Sign Language, version 1.3, 2011

[24] Rahimi A.: "An Erratum for 'A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition"', website of Ali Rahimi at MIT Media Laboratory, *http://xenia.media.mit.edu/~rahimi/rabiner/rabiner-errata/rabiner-errata.html*, 2000

[25] "Das Vokabelheft (Allgemeines Gebärdenwörterbuch)", online dictionary of German Sign Language at University of Hamburg, *http://www.sign-lang.uni-hamburg.de/alex/index.html*

[26] "Wikisign – DGS", online dictionary of German Sign Language, *http://dgs.wikisign.org/wiki/Spezial:Alle_Seiten*

[27] Official Microsoft Xbox website, introduction of Kinect, *http://www.xbox.com/en-US/kinect*

[28] "Countdown to Kinect: 17 Controller-Free Games Launch in November", Microsoft News Center, *https://www.microsoft.com/presspass/press/2010/oct10/10-18mskinectuspr.mspx*

[29] Kinect Fact Sheet, Microsoft News Center, June 2010, *http://www.microsoft.com/presspass/presskits/xbox/docs/KinectFS.docx*

[30] "Kinect Downgraded To Save Money, Can't Read Sign Language", News at Kotaku, *http://kotaku.com/5609840/kinect-dumbed-down-to-save-money-cant-read-sign-language*

[31] "CopyCat and Kinect", overview of the CopyCat Kinect demo on the website of the Center for Accessible Technology in Sign (CATS), *http://cats.gatech.edu/content/copycat-and-kinect*

[32] "PrimeSense Supplies 3-D-Sensing Technology to 'Project Natal' for Xbox 360", Microsoft News Center, *https://www.microsoft.com/Presspass/press/2010/mar10/03-31PrimeSensePR.mspx*

[33] "'Kinect for Xbox 360' is Official Name of Microsoft's Controller-Free Game Device", Microsoft News Center, *https://www.microsoft.com/Presspass/Features/2010/jun10/06-13KinectIntroduced.mspx*

[34] "The Open Kinect project - THE OK PRIZE - get $3,000 bounty for Kinect for Xbox 360 open source drivers", Adafruit Industries blog, *http://www.adafruit.com/blog/2010/11/04/the-open-kinect-project-the-ok-prize-get-1000-bounty-for-kinect-for-xbox-360-open-source-drivers/*

[35] "Bounty offered for open-source Kinect driver", CNET News, *http://news.cnet.com/8301-13772_3-20021836-52.html*

[36] "WE HAVE A WINNER - Open Kinect driver(s) released - Winner will use $3k for more hacking - PLUS an additional $2k goes to the EFF!", Adafruit Industries blog,

http://www.adafruit.com/blog/2010/11/10/we-have-a-winner-open-kinect-drivers-released-winner-will-use-3k-for-more-hacking-plus-an-additional-2k-goes-to-the-eff/

[37] "How The X-Box Kinect Tracks Your Moves", interview at National Public Radio with Shannon Loftis (Studio Manager at Microsoft Game Studios and Good Science Studios), Alex Kipman (Director of Incubation for Xbox at Microsoft), et al., *http://www.npr.org/2010/11/19/131447076/how-the-x-box-kinect-tracks-your-moves*

[38] "Windows Drivers for Kinect, Finally!", blog of Johnny Chung Lee, former employee at Microsoft, *http://procrastineering.blogspot.com/2011/02/windows-drivers-for-kinect.html*

[39] "History - OpenKinect", Wiki at OpenKinect.org, homepage of the libfreenect driver, *http://openkinect.org/wiki/History*

[40] "Roadmap - OpenKinect", Wiki at OpenKinect.org, homepage of the libfreenect driver, *http://openkinect.org/wiki/Roadmap*

[41] "PrimeSense™ Establishes the OpenNI™ Standard and Developers' Initiative to Bring the World of Natural Interaction™ to Life" - OpenNI News blog, *http://openni.org/news/5-primesense-establishes-the-openni-standard-and-developers-initiative-to-bring-the-world-of-natural-interaction-to-life*

[42] "NITE Middleware", introduction of NITE Middleware on the official PrimeSense website, *http://www.primesense.com/?p=515*

[43] "CopyCat", overview of the CopyCat platform on the website of the Center for Accessible Technology in Sign (CATS), *http://cats.gatech.edu/content/copycat*

[44] "American Sign Language Recognition using Kinect Skeleton features", demo video of the CopyCat Kinect demo on YouTube, *http://www.youtube.com/watch?v=qFH5rSzmgFE*

[45] "Microsoft Releases Kinect for Windows SDK Beta for Academics and Enthusiasts", Microsoft News Center, *https://www.microsoft.com/presspass/press/2011/jun11/06-16MSKinectSDKPR.mspx*

[46] "Multi-Gesture SDK for Kinect™/Asus Xtion Pro™/PrimeSensor™", official blog of Evoluce AG, *http://www.evoluceblog.com/2011/05/multi-gesture-sdk-for-kinect%E2%84%A2asus-xtion-pro%E2%84%*A2-primesensor%E2%84%A2/

[47] "Integrating Speech and Hearing Challenge Individuals", YouTube channel of Dr. Natheer Khasawneh, *http://www.youtube.com/user/knatheer#p/a/u/1/vVL398dUU5Q*

All websites were visited on September 27, 2011.