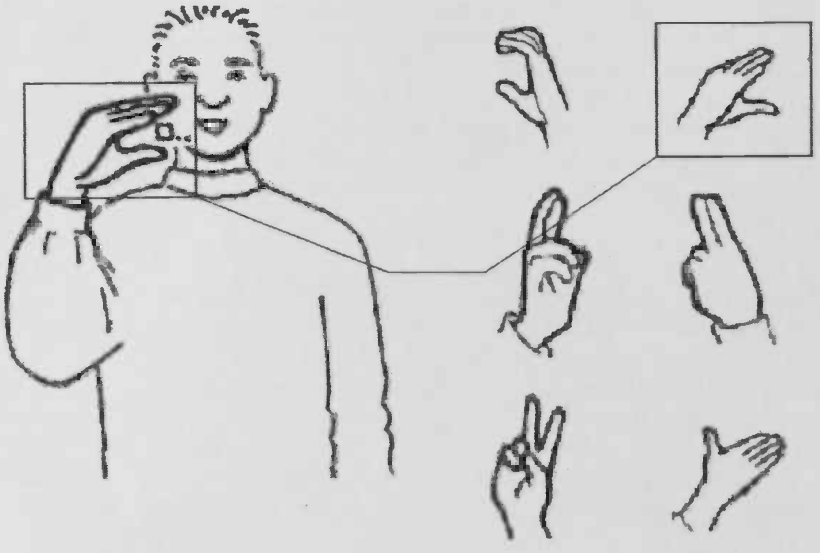
The Eye of the Beholder

Automatic recognition of Dutch sign language



Gineke ten Holt
1096516

September 9, 2004

Supervisors: Petra Hendriks (AI)
Tjeerd Andringa (AI)

Artificial Intelligence
University of Groningen

Preface

For as long as I can remember, I have been interested in languages. Sign languages especially fascinate me, because they are natural languages using a different modality: sign languages use images as the vehicle for information. This has strange implications: in sign language, it is easy to have a conversation with someone at the other side of a crowded room. On the other hand, it is impossible to talk to someone whose back is turned. It is such characteristics, such differences from spoken languages, that make sign languages so interesting to me.

I have studied Artificial Intelligence at the University of Groningen. I chose the direction 'Language- and Speech Technology'. For my final thesis, I wanted to set up my own research project at the university. My interest in sign languages motivated me to investigate the sign language side of a certain research area of Artificial Intelligence: the automatic processing of natural language. This area is also known as 'speech recognition', because it has mainly been investigating spoken languages. Investigating sign recognition is therefore interesting, because it could give us more insight into natural language processing: are techniques designed for speech recognition applicable to sign recognition, too? Or does the fact that they are designed for the characteristics of spoken languages mean that they are unsuitable for a language in a visual modality? The reverse is also interesting: can techniques developed for sign recognition be utilised by automatic speech recognition? I chose to work with Dutch sign language. To my knowledge, this is the first time automatic recognition has been investigated for Dutch sign language.

Apart from the scientific value, research into sign language recognition also serves a social purpose. Sign language users are minority groups practically everywhere, because usually less than 0.1% of a population is deaf. The majority of a population will therefore not speak sign language, which causes sign language communities to be somewhat isolated. So if techniques could be developed for the processing of natural sign languages, an application for interpreting sign language could be built, which would make communication between deaf and hearing easier.

Acknowledgements

The author would like to thank a number of people who provided assistance during this project, and without whom it could not have been completed. Firstly, I would like to thank dr. Richard Bowden of the School of Electronics and Physical Sciences, University of Surrey, for his assistance in processing Dutch sign language material. My thanks also goes to dr. Ella Bingham, of the Laboratory of Computer and Information Science, Helsinki University of Technology, who graciously answered my questions about Independent Component Analysis.

I thank Willem Bossenbroek for volunteering to record the Dutch sign language movie. Academy Minerva, the creative arts department of Hanze University Groningen, provided material for the movie background, for which I am also grateful.

I would also like to thank Pieter Zandbergen and the faculty of Psychology, Pedagogics and Social Sciences for providing the technical resources needed, Artificial Intelligence for providing the space in which the sign movie could be recorded, and drs. Ronald Zwaagstra, for providing a much-needed tape. Finally, of course, I thank my supervisors, dr. Petra Hendriks and dr. Tjeerd Andringa of Artificial Intelligence, whose advice and support were vital in every phase of this project.

Cover image: the sign 'to talk'. Used with permission of the Effatha Guyot Group, the Netherlands

Table of Contents

Chapter One: Introduction

1.1 Sign languages

For someone who can hear, it is difficult to imagine what it is like to be deaf. Many people associate 'deaf' with 'handicapped', but this is not really correct. A handicapped person, like someone who is blind, has trouble functioning independently in everyday life. Deaf people do not have this problem: they can ride a bike, drive a car, go to work, shop, and go on vacation like everybody else. The greatest problem for a deaf person is *communication*. Children learn language automatically by simply being exposed to it as babies. But someone who cannot hear as a baby cannot pick up a spoken language this way. Furthermore, for someone who was born deaf or became deaf early, a spoken language can never be a truly natural way of communicating. No matter how amazingly well some people can speak and read lips, it is impossible to participate fully in a spoken language when you cannot hear. With so-called 'lip-reading', only a part of the information can be picked up from the images (try for yourself to see the difference between 'tip' and 'dip', 'pit' and 'bit', etc.). But this does not mean that deaf children cannot learn language. They learn language in the same, natural way as hearing children, as long as it is in a modality that suits them: visual instead of auditory. Research shows that deaf babies that are exposed to sign language the same way that hearing babies are exposed to spoken language, learn this sign language in the same way, going through exactly the same phases (Emmorey, 2002). So language is not necessarily auditive: if sound is not an option, language can be developed and expressed in other ways. Note that a visual language is not the same thing as writing, though writing is also visual: written language is a visual *encoding* of a spoken language. To my knowledge, no language has ever developed in a purely written form only. But sign languages are an example of how a natural language *can* emerge in a visual modality.

For people who are prelingually deaf – that is, were born deaf or became deaf before they had learnt a language –, sign language is the language of choice, the only way of communicating easily and fluently, of expressing anything you want comfortably. Writing is not a viable alternative, which some people think, because it is not so much the inability to express and receive spoken words that troubles deaf people, as it is the inability to learn the spoken language in a natural way. Everyone knows that although you can learn a foreign language from a book, to *really* master it you have to speak it, hear it, be exposed to it. And mind you, in such a case, you already have the advantage of being proficient in one spoken language, your own, and of knowing what a language sounds like. Reading and writing in a language that is not familiar to you is no alternative for speaking your natural language, which for many prelingually deaf, especially those with deaf parents, is a sign language.
In the Netherlands, only one in every 1250 people is deaf (Koenen et al., 1993). Part of those have become deaf at a later age, so that spoken Dutch is still their mother tongue, and they can use lip-reading and written aids more easily. This means that deaf people, and especially the prelingually deaf, are a minority in the Netherlands. In the rest of the world, too, only about 0.1% of the population is deaf. Though there are exceptions – isolated communities with a high occurrence of hereditary deafness, such as the island of Martha's Vineyard (Groce, 1985) – usually only a small part of a community is deaf. This means that sign languages are minority languages, which are not shared by the larger part of the population, almost everywhere. As a result, sign communities are always rather isolated. This is true for groups using other minority languages, too, of course; the difference is, that speakers of *spoken* minority languages can very well learn the majority language; for deaf people, this is not an option.

Before we go on to the characteristics of sign languages, there is one important matter that must be mentioned. There is a variant of Dutch sign language called 'Nederlands met Gebaren' (Signed Dutch). This is a system of *encoding* spoken Dutch into signs. Similar systems exist for other sign languages (for example Signed English, which encodes American sign language; Liddell, 1980) The encoding can be done loosely, by simply substituting every word in a Dutch sentence with the most appropriate sign. A sentence like 'The cat sits on the roof' then becomes CAT SIT ON ROOF. Encoding can also be done strictly, by inventing signs for those grammatical modifications that are not used in sign languages, such as suffixes (like the *s* in sits), and using those 'grammar signs' to represent the original sentence exactly in signs. 'The cat sits on the roof' then becomes THE CAT SIT S ON THE ROOF. These 'signed' variants of spoken languages are not true sign languages: they are encodings in signs of spoken languages. This is an

important distinction: sign languages are natural languages with their own grammar; signed variants are encodings of spoken languages which use the grammar of that spoken language.

1.1.1 On the nature of sign languages

Sign languages are natural languages that emerge everywhere where groups of deaf people come into contact for some amount of time. Many sign languages therefore developed in and around educational institutes for the deaf. Sign languages are not constructed artificially. Nor are they mere gestural encodings of a spoken language, or systems of pantomime. They are complete, fully fledged natural languages, with their own grammars. In a sign language, everything can be expressed that is expressible in a spoken language: jokes, puns, philosophical discussions, poetry, sarcasm, everything. Furthermore, linguistic phenomena that have been observed in spoken languages, such as slips of the tongue, filler words such as 'er', etcetera, all seem to have their counterparts in sign languages (Emmorey, 2002). So sign languages seem totally equivalent to spoken languages, they simply use a different modality of expression: visual-gestural instead of auditory.
Because sign languages develop naturally, they are different in each country, and can even differ between regions within the same country. So just like there is Dutch, English, French, Chinese, etcetera, there is Dutch sign language, British sign language, French sign language, Chinese sign language, and so forth. Note that although Britain and the United States share a language, English, they have two different sign languages: British sign languages and American sign language. Not only does the grammar of a sign language differ from its spoken counterpart (that is, the grammar of Dutch sign language is different from that of Dutch, etc.), sign language grammars often also differ among themselves (Zeshan, 2004b). So the grammar of Dutch sign language can be quite different from that of Chinese sign language.
It is interesting to note, though, that there is a relationship between Irish, American, Spanish and Russian sign language, which all seem to be descended from French sign language (Deuchar, 1984). The reason is, that in the 18th century, the system of deaf education of abbé De l'Epée became famous. De l'Epée used the sign language of the French deaf in Paris in the education of his deaf students. A number of people used his techniques to start deaf institutes in their own country, thus exporting French signs to other countries. Dutch sign language, too, is descended from French this way (Koenen et al., 1993). There may be other relationships between sign languages, just like there are such 'family relationships' between spoken languages. But not much research has been done on this subject yet. Also, many sign languages have not been studied extensively yet, so that their precise grammar and rules are still unknown.

1.1.2 Parts of signs that carry meaning

In this section, I will discuss which parts of a sign determine its meaning. These are: *handshape, orientation of the hand, location of the hand, motion of the hand* and *non-manual component*. In Dutch sign language, if you change one of these aspects, you change the meaning of the sign, just like you change the meaning of a word when you change one of its letters. Every sign can be analysed in terms of these aspects: every sign has a handshape, palm orientation, location, motion (including 'none') and non-manual aspect (including 'neutral'). All these aspects can have various values: for instance, there are 70 different handshapes in Dutch sign language (Schermer et al., 1991). You could regard the aspects as phoneme groups, and the values of all aspects as phonemes, which fall into one of these groups. A sign is always made up of at least five phonemes: a handshape value, a palm orientation value, a location value, a motion value and a non-manual value. There can be more phonemes if one or more of these aspects change over time: all aspects can change within a sign.
Though it has not been proven formally, due to lack of comparative sign language research, it seems likely that these five aspects determine the meaning of a sign in all sign languages, as they do in Dutch sign language (Schermer, 1991). That is, there is no sign language known that regards handshape as insignificant for the meaning of a sign, or orientation, etc. Some languages may use fewer handshapes than others (like Adamrobe sign language, Nyst, 2004), but in no known sign language can handshape be just 'anything' and not affect meaning.
There is a point that needs addressing, though: human being have two hands. Are the four aspects that describe hand features important for both hands? The answer is: to some extent. Human beings have two hands, but one is always dominant (the right one for most people). In *one-handed signs*, only the aspects of the dominant hand matter, the other hand is ignored. In *two-handed signs*, both hands play a part. It has been proven, though, that in these signs, the non-dominant hand can only assume a limited range of values

for the aspects mentioned (Emmorey, 2002). For example: it can only have the same motion as the dominant hand (e.g. two hands moving forward), the mirror-image motion (two hands moving apart), alternating motion (when one goes forward, the other goes backward and vice versa), or no motion at all. Similar limitations have been found for handshape and orientation. Only location does not seem to be limited. The reason for the limitations must probably be sought in the heaviness of the cognitive load of managing two truly independent articulators: this is just too complicated. When the second articulator only does simple things, which require little extra attention, the load can be managed (Emmorey, 2002).
To give an example of sign aspects: in fig. 1 the sign UITDAGING (challenge) is shown. The five aspects are in this sign:

- handshape: fist.
- orientation: palm downward, fingers to the left (if they had been outstretched, they would have pointed to the left: this is how orientation is recorded).
- location: at the start: chest; later: neutral space in front of the body.
- motion: forward toward the listener.
- Non-manual aspect: neutral.

Every sign can be described in this manner. Note that this is a one-handed sign: the non-dominant hand does not matter here.

Figure 1: the sign UITDAGING (challenge). Used with the approval of the Koninklijke Effatha Guyot Groep, the Netherlands.

1.1.3 Finger-spelling and semantic marking

Apart from ordinary signs, sign languages use another way of transferring information: finger-spelling. Ordinary signs resemble words: they usually express one object, concept, action, quality, etcetera. But sign languages usually also possess a *hand alphabet*: a set of handshapes that each represent a single letter of the written form of the spoken language which is used in the environment of the sign language. So the Dutch sign language hand alphabet consists of 26 handshapes, each representing one letter of the Dutch (roman) alphabet. With this hand alphabet, words can be spelled out in handshapes. This technique is used to tell someone your name, or to communicate a word for which no sign is known. In the latter case, the sign language is *borrowing* a word from the spoken language to express a concept for which there is no native word. Borrowing is a linguistic phenomenon not limited to sign language (spoken languages borrow all the time, often eventually incorporating a word, which happened with 'computer' in many languages).
It is questionable whether finger-spelling is a natural part of sign languages, since it is an encoding (in signs) of an encoding (in letters) of a spoken language. However, Emmorey (2002) argued that since finger-spelling is picked up naturally by children, it should be considered part of the language. I believe, though, that in a signing community that did not co-exist with a speaking community, finger-spelling would

not emerge. I therefore find it doubtful to consider finger-spelling a true, natural part of sign languages. Finger-spelling is only used to spell out names and unknown words. But a letter sign is sometimes incorporated in a sign, for example by taking the first letter of the word (in spoken language) and using this as handshape in the sign. This phenomenon is probably an effect of the influence of the spoken language used in the environment on the sign language.
So there are two ways in which sign communication takes place: through ordinary signs and through finger-spelling. However, in sign languages, there are also *semantic markers*. These are used to convey semantic information, that is, information about a (part of a) sentence. Semantic marking is expressed through facial expression, eye gaze and body tilt. Examples are marking a sentence as a question, negation, and topicalisation (stressing the topic of a sentence). To give an example of the latter: in: "Broccoli I don't eat!", 'broccoli' is topicalised. The normal form of the sentence would be: "I don't eat broccoli". Topicalisation exists in spoken languages, too. Here it is often expressed through a change in pitch, duration, and volume.
Note that in sign language, facial expression can be used both as an aspect of a sign (the non-manual aspect), and as a semantic marker (such as raising the eyebrows to indicate a question). The difference is in the duration: non-manual aspects of a sign only last for the duration of the sign, but semantic markers last longer. Sometimes they last the entire sentence (this is the case for negation), sometimes they last only for as long as one sign, but even then, they remain for a moment after the sign is finished (this is the case for topicalisation), which distinguishes them from non-manual aspects of signs.

1.1.4 Context-dependent signs

In the previous section, I explained that certain aspects of a sign determine its meaning. There are morphological and syntactic processes, however, that can change certain aspects of a sign. (Morphology is forming different words from parts with a meaning of their own. Forming the plural of 'flower' by adding the 's' is an example: 'flower' denotes the object, 's' represents "plural", together they form the word 'flowers'.) Certain signs can get a different handshape, motion, orientation or location with different contexts. An example is the sign GEVEN (to give). This sign, a giving motion, starts at the location of the subject, and ends at the location of the indirect object. For this sign, direction of motion changes depending on who the subject and indirect object of the sentence are. Not all signs have these possibilities: some signs only have one possible form. But the signs that can vary usually have a 'citation' form (in which they are listed in a dictionary), and many 'inflected' forms. 'Inflexion' is put between quotes because it is not really the correct term: sometimes the variation represents conjugation, sometimes incorporation of a quality, sometimes other things. Not only verbs are capable of varying, other signs are, too. And not all verbs vary, some are fixed and have no variation. This subject will be discussed in more detail in section 4.4. For now, it is sufficient to note that many signs can have more than one form. This means that a sign cannot always be identified on the basis of all its aspects: sometimes, some aspects (e.g. handshape or motion) may be different from those in the dictionary because of context-dependency.

1.1.5 Movement epenthesis

Finally, there is one more phenomenon that I would like to discuss: movement epenthesis. In sign language, just like in speech, co-articulation occurs in continuous language. In speech, this causes sounds to change slightly under the influence of the sound that follows it, and the sound that precedes it. In sign language, this takes the form of sign parts such as handshape changing slightly under the influence of the next or previous part of the sign.
But in continuous sign language, another phenomenon exists which is called *movement epenthesis*. It is the occurrence of an extra movement in a sign stream caused by the fact that the position of the next sign differs from that of the previous sign. The hand has to move from the old position to the new one, and this causes and extra movement. For example, imagine a signer makes the signs MAN (man) and BOOS (angry) in succession. MAN is made in front of the forehead. But BOOS is made on the stomach. So after saying 'man', an extra movement is inserted from the forehead to the stomach. This has no meaning; it is simply caused by the fact that a hand cannot move instantaneously from one location to the other.
Movement epenthesis is a problem for continuous sign recognition, because it is difficult to detect when something is an epenthesis movement that should not be classified as a sign, and when it is an important (part of a) sign that carries meaning.

1.1.6 Areas of research

In this thesis, I study the automatic recognition of sign language. Most research into automatic sign recognition has focussed on processing finger-spelling or ordinary signs. Recognising semantic marking has not been attempted yet. Recognising a hand alphabet is usually manageable, because it simply consists of classifying a limited amount of shapes correctly. Recognising ordinary signs is a great challenge, though, because there is an infinite possible number of them, and it is not only the shape of the hand(s), but also motion, location, etcetera that determines meaning. Many researchers have tried to find ways of processing these ordinary signs. As I described in section 1.1.2, sign meaning is determined by the values of the five aspects of a sign. So to recognise a sign automatically, information on those aspects is necessary. How to obtain this information and what the process of automatic sign recognition entails exactly is the subject of the next section.

1.2 Automatic recognition of sign languages

Now that we have some insight into what determines the meaning of a sign, we can look at ways to recognise them automatically. I will first describe the various steps of the automatic sign recognition process. Then I will address some issues that all methods aspiring to process natural language have to deal with.

1.2.1 The sign recognition process

How does sign recognition work? Broadly, it is a process of four steps:

1. Data acquisition
2. Data encoding
3. Classification
4. Translation

This process is shown schematically in fig. 2. I will address each step.

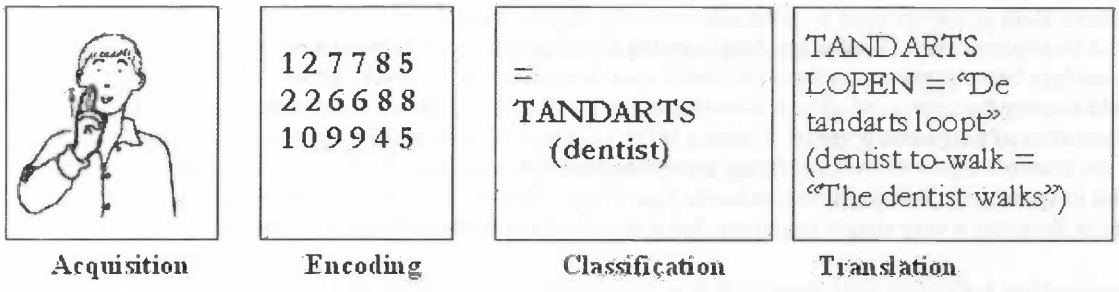


Figure 2: Schematic representation of the sign recognition process. First, a sign has to be captured somehow. These data have to be encoded in some form. Then a classification has to be made: which sign is it? And finally, translation is needed when a sign sentence is to be represented in some other (spoken) language. As yet there are no feedback channels. (Sign image used with the permission of the Effatha Guyot Group, the Netherlands)

Data acquisition

First, signs have to be captured in some way. The ways to do this fall into two categories: with cameras, and with sensors. Image based methods record signs with a camera, or a stereo camera, or two cameras, or

a camera and a mirror, and so forth. In any case, they all capture data in images, which have to be processed later.
Sensor-based methods do not use cameras, but use sensors to capture certain data that provides information about a sign. The most common setup is a combination of instrumented gloves and magnetic trackers. Instrumented gloves are gloves containing sensors that can measure the amount of bend of certain joints, and sometimes also the amount of rotation. Such data indirectly provide information on handshape and hand orientation. Magnetic trackers are sensors that calculate their position, and sometimes their orientation, relative to a fixed source. By putting magnetic trackers on the wrists, information on the position of the hands can be obtained. And when the fixed source is worn on the signer's back, the reported position of a hand becomes 'position relative to the signer'. In some projects, only magnetic trackers are used for data acquisition, but most use both gloves and trackers.

Data encoding

After the data have been captured, the information that is useful has to be extracted and encoded in some form. This step does not really exist for sensor-based methods, since these receive only information that is useful (why would you add a sensor that reports information on something you are not going to use?). But for image-based methods, data encoding is important. There is much information in a sign movie that is unimportant: the background, the colour of the signer's clothing, etcetera. What you *want* to extract is information on handshape, hand position, orientation, motion, and non-manual features: the sign aspects described in the previous section. Extracting this information and encoding it in some form is the 'data encoding' step. There are many possible ways of doing this: for example, you could track the hand and encode a sign by annotating x- and y-position of the hand and size of the hand for each frame. This would give you information on position and motion, and some indication of handshape (through the size of the hand). But it would not give you information on hand orientation or non-manual features, so this manner of encoding would probably not be good enough. Still, it should illustrate what data encoding entails.

Classification

When the important information has been extracted from the data, a classification must be made on the basis of it. That is, now that we have information on handshape, location etcetera, we must decide which sign was made. This usually entails learning a number of signs and then classifying an unknown sign as one of these. Both processes must be done automatically. Again, there are many ways in which classification could be implemented. Usually, machine learning techniques of classification are employed. Sometimes, researchers borrow techniques from automatic speech recognition also. An example of classification: you could employ the nearest-neighbour classification algorithm. For this you would have to pack all information of a sign into a vector. Training is then a matter of marking those points in space represented by the example sign vectors. Classifying entails packing the unknown sign into a vector, too, marking its point in space, and checking which example sign it lies closest to. The closest sign is the sign recognised. This is of course a very simple technique, but it should illustrate the classification process.

Translation

The final step of the classification process is usually ignored at the moment. And if you only want to recognise isolated signs, then it is not really necessary. But if you want to translate natural sign language, a final step is needed. Simply classifying each sign and outputting the result is not enough to translate a sentence: word-for-word translation is only understandable for simple sentences. So if the goal of the recognition process is a true transcription (in some spoken language) of what was said in the sign language, then the classification results have to be transformed into a well-formed sentence somehow. How to do this is still a difficult question at the moment.
But as we shall see later (section 4.4), translating is not something that is easily added at the end of the recognition process. Sign languages employ space to express certain syntactic structures. So if syntactic information is to be used later to reconstruct a sign sentence, spacial syntactic information has to be detected straight away at the acquisition/encoding stage, else the information is lost. This means that translating is not really a fourth step: it should be incorporated into every step of the process.

1.2.2 Issues for automatic recognition of natural language

For all methods that aim at performing automatic recognition of a natural language, certain choices have to be made. This subject will be dealt with more extensively in chapter 2, but here, I want to mention a few of the most important issues, to give an idea of the variability that is possible in creating an automatic language recognition system.

1. **Continuity**. Should the system be capable of processing continuous speech, or is it sufficient to identify isolated words?
2. **Constraints on grammar**. If the system *is* to handle continuous speech, is it required to be capable of understanding every legal utterance of a language, or only utterances that obey strict rules of word order?
3. **Size of the vocabulary**. How many words is the system required to understand? Is a small vocabulary sufficient, or must it be able to recognise every word in the language?
4. **Real-time performance**. Is the system required to work as fast as a person speaks, or can it have more time to process the language?
5. **Signer-independence**. Is the system required to understand any person, or does it only have to deal with one specific speaker?

Ideally, a language recognition system would be able to recognise continuous, unconstrained natural language from any person using whatever words they like in real-time. However, each of these issues makes the problem harder: it is easier to process isolated words than continuous speech, easier to process sentences from which the form is known in advance than sentences that can have any form, etcetera. So a trade-off usually has to be made between usefulness and performance.

1.3 Project overview

In this project, I want to investigate the automatic recognition of Dutch sign language. To my knowledge, this is the first project that tries to achieve automatic Dutch sign recognition. I want to investigate the current state of sign recognition research, and apply the best method found to Dutch sign language.

So my research question is:

What is the best way to automatically recognise Dutch sign language?

This comprises the following sub-questions:

- Which methods of automatic sign language recognition are there?
- Which one is the best?
- How can I implement part of this method?

To answer the first two points, I will conduct a survey on sign language recognition research. I will determine which criteria a sign recognition method should meet, and evaluate the existing methods with these criteria to find the best one.
Then, I will implement a part of this method. I cannot implement an entire method, because that would take too much time. Instead, I opt to implement only the classification part of a method. To do this, I need pre-processed data, data for which the encoding step of the recognition process has already been performed. I choose the to implement classification, because this is the most interesting part from my point of view. Encoding data is more of a computer vision problem if it concerns image data, and is no problem at all if it concerns sensor data. Classification is a problem of cognitive science/modelling, and therefore more interesting to me.

I will start by presenting my survey on sign recognition research in chapter 2. I will explain the criteria that will be used to evaluate the sign recognition methods, describe the various methods, and choose a method to implement. This implementation will be described in chapter 3. In chapter 4 the results of the project will be discussed and chapter 5 will present the conclusions.

1.4 Conventions of notation

In the text, a sign is expressed as a gloss: a word representing the meaning of the sign. The glosses are in capital letters. They are in Dutch and are followed by an English translation between parentheses. So the sign for 'to go', 'gaan' in Dutch, would be expressed as: GAAN (to go).
In this paper, the words 'speech', 'to speak', 'speaker' and 'listener' will be used for both sign language and spoken languages: 'speech' is used in its meaning of 'language utterance', regardless of the modality of that language. Usually it will be clear from context whether spoken or signed language is meant, or else it is language in general that is meant and the distinction is unimportant. The terms 'signing', 'to sign' and 'signer' will also be used.
The terms 'segmenting speech' and 'segmenting an image' will be used several times. It is important to note the difference of both uses of 'to segment': in the first case, it indicates 'dividing continuous speech into words'. The second case has to do with dividing an image into 'parts' such as a hand, a face, a shoulder line, etcetera. So 'segmenting an image' means finding certain important objects in an image.
The verb 'to implement' is used mainly in its interpretation of 'to implement as a computer program'. If another interpretation is meant, it will be mentioned explicitly.
As we have seen in this introduction already, the terms 'aspect', 'feature', 'parameter' and '(parameter) value' will be used a lot. It is important to clarify what is meant by each. An *aspect* of a sign is a part of it that bears meaning: handshape, hand orientation, location, motion and non-manual aspect. These could be used as *parameters* in the sign encoding stage of the recognition process, though a method can also choose other parameters. Parameters are the characteristics of a sign that a method extracts and encodes. These parameters can have certain *values*: each parameter has a number of possible values. For instance the parameter 'handshape' could have the values 'fist', 'flat hand', etcetera. And the parameter values are all *features* of a sign, a feature being simply a characteristic.
'Location' and 'position' are used interchangeably when describing an aspect of a sign; the same goes for 'movement' and 'motion'.
Finally, there is the distinction 'sign' versus 'gesture' which is important: a sign is part of a sign language and is defined strictly. A gesture is a (hand) motion not part of a sign language, but used by both signers and speakers. Gestures can vary a lot more in appearance than signs, and their meaning is not as well defined. An example of a gesture is the 'throw-away' motion you make when you have had enough of something.

Chapter Two: Sign Recognition Methods

2.1 Introduction

The subject of this project is the automatic recognition of Dutch sign language. To determine the best way to recognise sign language automatically, it is necessary to review the methods that have been developed and their results. This will help to determine the best way to handle the problem of Dutch sign recognition. In this section, I will discuss several automatic sign recognition techniques. After a few introductory remarks, I will first explain the criteria that will be used to evaluate the methods. After that, I will discuss the current methods. I will end by summarizing the results of my study and concluding which is the best method for recognising Dutch sign language. This method, and its application to Dutch sign language, will be described in the next chapter.

2.1.1 Finger-spelling

Automatic sign recognition is a relatively young field of research: only since the 1990s has there been serious research in this area. There are some projects studying automatic recognition of finger-spelling that are older, but I do not consider recognition of finger-spelling real sign language recognition. The practical problems of recognising finger-spelling are very different from those of recognising signs: in finger-spelling there is only a fixed number of hand shapes that needs to be recognised. Orientation and motion are only marginally important and location is always the same. Non-manual components can be ignored completely. This makes the problem much simpler than that of sign recognition. It becomes more complicated when finger-spelling has to be recognised in between natural signs, that is, when the data consist of natural sign language with some finger-spelled words among it. Because then, a recogniser has to detect whether something *is* a finger-spelled word, or whether it is (part of) a sign, and process only the finger-spelled data. But to my knowledge, none of the researchers studying finger-spelling has tried such an approach. When it is certain that the data the recogniser receives consist only of finger-spelling, the recognition process becomes relatively simple, because a lot of sign aspects can be ignored (motion, orientation, non-manual components, location). I will therefore not discuss automatic finger-spelling recognition here, but only concern myself with sign recognition.

2.1.2 Do different sign languages need different methods?

Before I start reviewing existing methods, there is one point that must be addressed. The purpose of my project is to find and implement the best method for recognising Dutch sign language. Does it matter, though, whether it is Dutch sign language, or American, or British, or Pakistani etc.? Is a method that works brilliantly for Korean sign language automatically applicable to Dutch sign language? The answer is, that it is not certain. There have not been many studies comparing the characteristics of sign languages around the world, and the lack of written sources makes research into the kinship of sign languages extra difficult. So we do not yet know which characteristics, if any, are universal to sign languages. It is clear, however, that there is a lot of variation across sign languages (Zeshan, 2004a, 2004b), but this variation mainly concerns the way certain syntactic constructions are expressed in different sign languages. The parts of a sign that determine its meaning – location, motion, orientation, handshape and non-manual component – seem universal to all sign languages. That is, these aspects seem important for the meaning of a sign in all sign languages: there is no known sign languages that regards for example handshape as totally insignificant to the meaning of a sign. The fact that phonology and morphology have not been studied extensively for many sign languages makes it difficult to draw conclusions in this area.
My approach has been to review all methods according to the criteria I will specify in the next section. Whether such a method is applicable to Dutch sign language depends on which parts of the data it uses to recognise signs. If it uses parts that are key in determining the meaning of Dutch sign language signs, then there is no reason why the method should not work as well for Dutch sign language as it does for the sign language it was designed for. However, the problem of differences between sign languages is an important one to keep in mind.

2.2 Criteria for reviewing methods

To review existing methods of sign recognition critically, criteria must first be determined. What makes a method successful? When can a method be considered good enough? And what makes a method more or less promising for the future? In this section, I will describe the criteria that will be used to evaluate recognition methods. These criteria fall into two categories: criteria concerning general issues, which all sign recognition methods deal with, and criteria having to do with issues that depend on the purpose of this particular research project. Then there is one other criterion that does not really fit into either category and will therefore be discussed separately.

2.2.1 Purpose of this research project

Automatic sign language recognition research can be done for different purposes. For instance, it can be used as part of general gesture recognition research (for the difference between signs and gestures, see section 1.4). Because signs are more structured than gestures, sign recognition is a good starting point for research into gesture based interfaces or virtual reality. However, the purpose of this research project is to contribute to the eventual construction of an automatic sign language interpreter. Such a system would be useful for making communication between deaf and hearing easier, for example at airports or post offices, but also in social contexts. And if an automatic speech-to-sign system could be constructed also (an area in which there is also a lot of research going on), a complete sign language interpreter could be built. Such a system would not easily be perfected, so that human interpreters would still have to be used in important situations, such as a consult with a doctor, a lawyer or a notary. Nevertheless, the automatic interpreter would be very useful in more day-to-day and social contexts.
To be able to handle natural sign language, a recognition method would have to satisfy certain demands. Because the (eventual) purpose of this research project is interpretation, recognition methods will be reviewed according to these demands, too. They will be described in the section after the next. First, general criteria will be discussed.

2.2.2 General criteria of usefulness

All methods that perform automatic sign recognition have to perform certain tasks that are inherent to dealing with (sign) languages. In an overview, they are:

- collecting and using all aspects relevant to the meaning of a sign
- extracting features from the data
- dealing with expansion of the lexicon
- dealing with intra-signer differences in signs
- dealing with inter-signer differences in signs

Specific for methods that collect data visually, are:

- segmenting the image (finding hands, face, etcetera)
- extracting appropriate features from the images (how to conclude which handshape was made, which motion, etc., on the basis of the segmented image)

Specific for methods that collect data with instrumented gloves, are:

- collecting data on non-manual features

How well methods are suited to perform these tasks determines how good they are. Apart from these issues, there is another important criterion: how well expandable a method is. Methods might not meet all these criteria right now. But some methods could well satisfy them in the future, with faster computers, or added modules, or better computer vision methods. Other methods are dead ends: it is not clear how extra parts could be added, or how a faster computer would help to improve their results. So expandability is an important criterion.

Finally, it would be a big advantage if a method were suitable for different sign languages. Its suitability would depend, as we have seen in the section 2.1, on how much different sign languages have in common. But the more generally applicable a method seems, the better. Because this would mean that all the efforts

made to achieve recognition of one sign language do not have to be made again for each new language. This would of course make a method very attractive.

So the general criteria for a sign recognition method are the following:

1. Is it capable of collecting and using all aspects of a sign that determine meaning?
2. Do the features it extracts from the data sufficiently represent all these aspects?
3. Can it deal with additions to the vocabulary?
4. Is it capable of generalising over differences between instances of the same sign?
5. Is it capable of generalising over differences between signs made by different signers?
6. Is it expandable? Could new modules be added to improve its performance?
7. Is it applicable to different sign languages?

I will review them briefly.

1. Collecting all aspects

The first criterion is rather broad. It encompasses the problems of collecting data on all aspects of a sign – which is difficult for glove-based methods, because they have no facial information available –, and finding these aspects within the data. This is difficult for image-based methods because they have to find hands and face and extract the relevant features, such as handshape and orientation, from the images. For glove-based methods, it is easier: since they work with sensors, they have already 'found' the hands and their position, orientation and amount of finger-bend. But they still have to infer which handshape is being made and which motion, so glove-based methods, too, have to find the aspects of a sign in their data.

2. Features representing all these aspects

The second addresses the issue that all relevant aspects of a sign must be represented by the features extracted: otherwise, some signs will be indistinguishable for a method. If a method extracts only hand information, such as handshape, location, etc., it will not be able to distinguish between signs that differ only in their non-manual aspects.

3. Adding to the vocabulary

The third criterion concerns the problem of adding new signs: if a method only works for a very small vocabulary, or if it is only capable of training on an entire data set, additions to the vocabulary will cause great problems. For example: imagine that a neural network was used for sign recognition. It would be trained on a set of signs, and, if all went well, consequently be able to classify these signs correctly. If a new sign were added, though, the whole training process would have to start all over again, because neural networks are inflexible: they learn to classify a certain data set, but cannot add another classification item to their existing layout without the risk of forgetting older items. So they have to start learning the new, enlarged data set as if it were an entirely new set. Such a system would not satisfy the third criterion.

4. & 5. Generalising over sign instances and different signers

The fourth and fifth criterion are about ability to generalise. A method has to recognise a *sign*, not a certain instance of a sign. So if it can only recognise a sign if it is made in the exact same way as during training, it has not really learnt the sign, it has learnt one instance of it. The nature of language is such, that signs and words are often performed slightly differently by different speakers, and even by the same speaker at different times. A method must be able to deal with these variations. Otherwise, it would not be recognising signs, just instances.

6. Expandability

The sixth criterion deals with the expandability of a system, as described earlier in the section. The easier extra modules can be added and new algorithms used, the better a system can be improved in the future.

7. Applicability to different sign languages

Finally, the seventh criterion addresses the issue of usability of a method for different sign languages, also described above. If a method is not suitable for other sign languages without starting over again from the beginning, it is less attractive than when its results could in part be applied to other sign languages as well.

2.2.3 Criteria specific to the purpose of the project

The purpose of this research project, to build an automatic interpreter of natural sign language, brings its own criteria with it. If a method is to interpret natural sign language, it must be capable of handling natural signed speech in normal environments. This means:

- dealing with large vocabularies
- dealing with continuous signed speech
- working in real-time
- working with cluttered backgrounds
- working without too much hardware attached to the signer

Apart from this, a method has to be successful enough. What is enough? A method that recognises only 30% of the signs correctly is worthless: it would classify two out of every three signs wrongly. It would be impossible to understand what was said under such circumstances. In speech recognition, a recognition percentage of 95% is considered minimal (according to T. Andringa, personal communication, Jan 2004). On the other hand, an interpreting application does not have to work perfectly to be useful: even with imperfect recognition, such an application would already be useful in aiding communication between hearing and deaf, especially in face-to-face situations where clarification can be asked if something is unclear. But of course, the higher the recognition rate, the better.

So the criteria specific for a research project that has the aim of building a sign language interpretation system are:

1. How high is the recognition percentage of a method?
2. Can it handle large vocabularies?
3. Can it handle continuous signed speech?
4. Can it work in real-time?
5. Can it work against all kinds of backgrounds and clothes worn by the signer?
6. Can it work without the signer wearing hardware and wires to power sockets?

I will review a few of these criteria here. The second criterion is about the problems that can arise when large vocabularies are used. Firstly, a method has to make fine distinctions in that case: crude classifications sometimes work if the vocabulary is small and the signs in it are very different. But if a vocabulary is large enough, there will be many signs that only differ in small aspects. If a method cannot make these fine distinctions, it cannot deal with large vocabularies. Secondly, if a method uses models for whole signs, instead of models for parts of signs, it will end up with a huge amount of models in the case of a large vocabulary. In the recognition process, it will have to search through all these models. To avoid this becoming too time- and resource-consuming, it will need methods to deal with this.
The third criterion deals with the problem of continuous speech: it is difficult to find word-boundaries in continuous (signed) speech, and signs change under the influence of the preceding and following sign (co-articulation). If a method cannot deal with these problems, it cannot handle natural sign language.
The fifth criterion addresses a problem for image-based methods: segmentation. Finding and tracking hands and face is often much easier under controlled circumstances: a neutral background, dark clothing, etc. To work as an interpreter, though, a sign recognition system will have to work in the real world. This means working against possibly interfering backgrounds. A method must be able to deal with this kind of interference to be able to work in the real world.
The last criterion handles the problem of hardware: glove-based methods require the signer to wear instrumented gloves and magnetic trackers. It is debatable whether this is a real problem, as long as all the equipment is wireless. But it would still be cumbersome, so the less hardware a signer has to wear, the better.

2.2.4 Criterion of cognitive plausibility

Apart from criteria inherent to sign language processing and criteria specific to the purpose of this project, there is a criterion fitting in neither class, which I will discuss here: the criterion of cognitive plausibility. This criterion addresses how well the methodology of a certain approach matches the human methodology.

That is: does the system process sign language and classify signs in the same manner that human beings do? Or does it work entirely differently? An example of a cognitively plausible method is a method that works with a camera in front of the signer. This is the normal viewpoint for a human being listening to sign language. Such a method finds hands and facial expression in the images and determines which sign was made on the basis of this information. Less cognitively plausible is a camera mounted on a cap, which looks straight down onto the signer's hands. This is not a normal viewpoint for a listener. Also not very plausible is a method using instrumented gloves. Such a method uses amount of finger joint bend as its data, and this is information that is not directly accessible to human beings receiving sign language: at most, they can infer it from the hand images. So cognitively plausible means, broadly speaking: what humans do.

So why is a method better when it is more cognitively plausible? Human beings may not be able to look straight down onto the hands of the speaker, but if it aids the recognition process, what does it matter? The problem is, that speech is a communication process, a process of exchanging information, always performed between two (or more) human beings. A speaker is used to having the listener approximately opposite him, not straight above him or measuring the amount of bend of his finger joints. So he knows what the listener sees, and what he does not. Even if this is not conscious knowledge, it will have its effects: a signer knows – or has become used to the fact, if you will – that a listener does not see those fingers obscured by other fingers in a certain handshape or a certain sign. So it does not matter which position those fingers assume: he knows that the listener cannot see them. And even if the listener happened to see them, in a mirror for example, the listener would not conclude anything on the basis of those data, because *he* knows that the speaker would never transmit information that way, because ordinarily the listener would not be able to receive those data. Both speaker and listener have become used to the fact that some parts of signs will be visible, and can therefore be used to transmit information, and some parts are not visible, and will therefore not be information-carriers. To use information different from that which the listener receives, such as images as seen from straight above the hands, or finger-bend sensor data, is dangerous, because a system can then associate meaning with aspects that are not meaningful, such as obscured fingers, small details not detectable from normal signer-speaker distance, etcetera. A signer only transmits information with those parts of his signs that he knows can be distinguished by the listener, so the best way to understand him is to make use of those same parts, no more and no less.

In practise, the unseen parts of a sign may be constant for signs, because performing a sign the same way every time, including position of obscured fingers, is the most natural thing to do. But whether this is true for different signers is another matter already: some signers may curl up their obscured thumb in a certain sign, some may prefer to leave it straight. It does not matter, because it cannot be seen anyway, but to a system based on finger-bend, it does make a difference. And how should such a system distinguish between cases such as this, where the amount of bend in the thumb does not matter, and cases where the thumb *is* visible, so that the amount of thumb-bend is very significant indeed? This is why a more cognitively plausible method is preferable. This criterion does not weigh heavily in the determination of the best method, though.

Now that the criteria have been determined, we can look at the current methods in the field of automatic sign language recognition. I will describe my findings in the next section.

2.3 Sign recognition methods

In this section, I will describe various methods currently used to perform automatic sign language recognition. The descriptions will be brief, the emphasis will be on assessing the usefulness of each method. I will review each method according to the criteria specified in the previous section.

2.3.1 Glove-based methods

I will first review the methods that are glove-based. These methods use instrumented gloves to measure the amount of bend in a certain number of finger joints. They generally also use magnetic trackers on the wrist(s) to determine the position of the hand(s). Some use only magnetic trackers, and no gloves, but since most use gloves, I have chosen to call this group 'glove-based', which is more informative than a name such as 'sensor-based'.

2.3.1.1 Machine learning

In machine learning, researchers try to develop algorithms that allow machines to modify their own behaviour while trying to achieve a goal. Instance based learning (IBL) is an example of such an algorithm: it classifies objects according to their nearest neighbour. Training examples of objects (one per object) are placed into 'object space'. A new object is compared with all these examples. It is assigned the class of the object it resembles most closely, according to some measure of similarity. In this manner, objects can be automatically classified, as long as one training example of each class has been used in the training phase. Another example of machine classification is decision-tree building. This is a hierarchy of decisions based on attributes. For instance, to classify a bird, we would first look at its attribute "flight". If this is a "no", then a lot of birds can be discarded. Next, we look at its attribute "beak shape". If this is sharp, it leaves only a certain group of birds, if it is blunt, broad, flat, etc. it leaves other groups. Within these groups, the next attribute is examined, etc. Such a decision tree can be built automatically on the basis of certain examples of attributes and classification: for instance the attributes of a penguin and the fact that it is classified as a sea-bird, the attributes of a bluetit and the fact that it is classified as a seed-eating bird, etcetera. The machine finds the best tree according to the examples and then uses this tree to classify new, unknown objects according to their attributes.

Kadous (1996) used instance based learning (IBL) and decision-tree building to automatically classify signs of Australian sign language (Auslan). He used a simple instrumented glove to capture data. The data he used were position of the hand, bend of the first four fingers (no little finger), and wrist rotation, all of which were captured in very crude categories. He used only one glove. He extracted the following features from the data: histograms for x, y and z position, histograms and time segment averages for finger bend and rotation. Histograms were made by dividing the range of possible values of a variable, such as x position, into categories and calculating the relative time the variable was within that range of values. For instance, x-position range is divided into two segments. It then turns out to be for example in the lower half for 70% of the time, and in the higher half for 30% of the time. Time segment averages are the average values of variables during certain time segments. The number of segments was determined empirically in both cases, that is, the best number of segments was found by trial-and-error.
Kadous tested his method on a 95-sign vocabulary. He used five signers, and obtained between 8 and 20 samples per sign from each of them. Kadous trained and tested per signer. He achieved results of about 80% using IBL, and about 40% using decision-tree building. When he trained his system on four signers and tested it on a fifth, the recognition rate was 15%.

Reviewing this method with my criteria, some serious problems can be found. The recognition rate is reasonable, 80% for a 95-sign vocabulary (a perplexity of 0.011). But the features that are collected are hardly exhaustive: not only is there no facial information, which is the case for all glove-based methods, but there is also no information on the little finger, an important finger in sign languages. Furthermore, the crude categories in which the features were classified would probably not be sufficient to deal with fine distinctions in signs that are rather alike. But these defects could be repaired by using another glove (better gloves exist). There is no practical problem with adding signs to the vocabulary. It is simply a matter of adding a training example of the new sign in the 'object space' of the nearest neighbour classifier. With

good results on 8 to 20 samples per sign, the system seems capable of generalising over intra-signer variability, but inter-signer variability is a problem: performance drops dramatically when the system is tested on an unseen signer. The features extracted are probably significant in signs from other sign languages as well (they are significant for Dutch sign language, at least), so this method could probably be applied to different sign languages.
Since this system only classifies single words, of which start and end have to be artificially indicated, not much can be said about its capability of handling continuous sign language. Large vocabularies could become a problem if more features are not extracted, but the system's recognition time was shown to be $O(\log n)$, with **n** the number of signs in the vocabulary. This means that recognition could be performed in real-time on a large vocabulary, though whether this would still be so after the extra features were added, is of course unsure. Finally, the signer has to wear hardware, and measuring finger-bend etc. is not very cognitively plausible. These two criteria are a problem for all glove based methods.

Kadous's system is one of the earliest attempts at automatic sign language recognition. Some other approaches will be discussed next.

2.3.1.2 Artificial neural networks

Artificial neural networks (ANNs) are networks of connected nodes. These nodes are organized in layers. Each network has an input layer, an output layer and any number of hidden layers (including zero). There can be any number of nodes in a layer. All nodes can be connected to all other nodes in neighbouring layers, but fewer connections are also possible. Some networks only allow transitions to the next layer (feed-forward networks), some also allow backwards transitions. A connection has a certain weight. This means that the activation of a certain node is transmitted to the next node according to the weight of the transition between them. A node receives values from other nodes in different strengths. The values and their strengths determine which value the receiving node gets. And then it transmits this value again to all nodes it is connected to according to the weights of those connections, which results in certain values in the next nodes, etcetera, until the output layer is reached.
A neural network receives an input on its input nodes, and delivers an output on its output nodes. It can learn to produce the correct output after a certain input by adjusting the weights of its connections. It does this according to its learning rule, which can be simple or quite complex.
The great advantage of ANNs is, that they can learn an input-output connection without having to be taught *how* to do it. The network simply iterates through the data over and over until its weights are such that each input produces the correct output. The disadvantages are that a network needs a lot of iterations and time to learn a combination, and that a network cannot add something to its knowledge without the risk of forgetting other parts. Which means that for each additional input-output combination it must handle, it has to learn the whole set anew. So ANNs are self-learning, but inflexible. For more information, see for example Gurney (1997).

Vamplew & Adams (1998) attempted to use neural networks for the recognition of Australian sign language (Auslan). They created separate networks for the recognition of handshape, orientation, location and motion. They used an instrumented glove with 18 sensors to capture data of one hand, and a magnetic tracker to determine position and orientation with respect to a fixed source. They used different combinations of these sensors as inputs for each of their networks. The first three networks were feed-forward nets, trained by adjusting the weights to obtain a certain output for each input, as described above. The motion network was more complicated, because motion is temporal in nature. They tried to use a recurrent network so as to preserve the previous state of the net and use it in processing the next time-step. It turned out, though, that it was better to pre-process the whole data sequence of a sign and extract the information useful for the analysis of motion, such as the sum of all differences in position along every axis, the total amount of movement on every axis, and the sum of all velocities. These parameters were then fed into a regular feed-forward network to classify the motion.
The authors used the networks to determine handshape, orientation and location for both the start and the end of a data sequence (one sign). They determined one motion value for the whole sequence. On the basis of these seven features, a classification was made. The authors did not use a neural network for this classification, because it would have to be very large and it would have to be trained anew for each addition to the vocabulary. Instead, they used two classification algorithms: a nearest neighbour lookup and the C4.5

inductive learning system. The first classifies according to the similarity of its test case to the examples it has received of each sign. It performed better when it used definitions from a dictionary as examples than when it used the image sequences from the training data. This is possibly because training examples are instances, and a description is more generalised.
The C4.5 inductive learning system is a tree building algorithm which creates a decision tree for the data. This building can be done very quickly, so it is no problem to do it again each time the vocabulary should be enlarged.
Data were collected from seven signers for training and testing, and from three others for testing only. The authors used a 52-word vocabulary, and only one instance per word per signer, so 52 signs were collected from each signer. The nearest neighbour classification produced the best results: a recognition rate of about 94% for the signers it had trained on, and 85% for the unknown signers.

The test results of this method are quite good: 94% for known signers, and 85% for unknown signers (perplexity 0.019). The system uses almost all significant features of signs, only non-manual components are ignored. Because of the modular nature of the system, however, a face recognition network (or other face recognition device) could easily be added. This would simply mean that the final nearest neighbour classification would receive 9 values instead of 7 to classify with. So this system is expandable. How fast the networks are, is uncertain, so it is not clear whether the entire system can work in real-time. But if they are fast enough, the system should be able to deal with large vocabularies, since the nearest neighbour classifier works in reasonable time even with large data sets. Adding to the vocabulary is also not a problem: an example can easily be added to the nearest neighbour classifier, and the feature detection networks do not have to be trained again: as long as they have been trained on a divers enough data set, they can detect all possible features. Because, although the number of possible signs is endless, the number of sign features is limited (just as the number of possible words is endless, but the number of sounds in a language is limited).
But there are important criteria which this method does not meet. It can only process single signs, and then only when start and end are indicated with a hand-held button. Furthermore, it is not easy to see how this could be remedied, because the system needs an entire sign to be able to classify: it must determine values for start and end, and pre-process motion over the entire duration of a sign. How such a method could deal with continuous sign language is not clear. Also, when tested on three unknown signers, the recognition rate of the system drops rather sharply with 9%. This could drop even further when more different signers are used. So whether it can generalise well over many signers is not entirely certain. And finally, as is the case for all glove-based methods, the signer has to wear hardware on his body, and the approach is not very cognitively plausible.

2.3.1.3 Hidden Markov models

A hidden Markov model (HMM) is a model consisting of states, outputs and transitions. At each time-step, the model is in a certain state. This state can produce several outputs with varying probabilities. From a state, a model can go to several other states in the next time-step, also with varying probabilities. This means that an HMM can produce a certain output sequence with a certain probability. A simplified example is given in fig. 3. This model can produce the sequence 'RVS' with a probability of 0.1 * 0.75 * 0.5 * 0.9 * 0.7 * 0.5 = 0.01, or 1%. HMMs can be trained to represent certain elements, such as a phoneme, or a word. This training does require a lot of training samples. If trained successfully, the model that has the highest probability of producing the observation sequence of a test word, phoneme etc., is the model representing that test word, phoneme etc. In other words, when a bank of sign models has been trained, and a certain unknown sign is observed, the model that has the greatest chance of producing the observations of that unknown sign is the model that represents the sign that was made. HMMs are used widely in speech recognition, and have also been applied to sign recognition by several researchers. For more information on HMMs, I refer to Rabiner (1993).

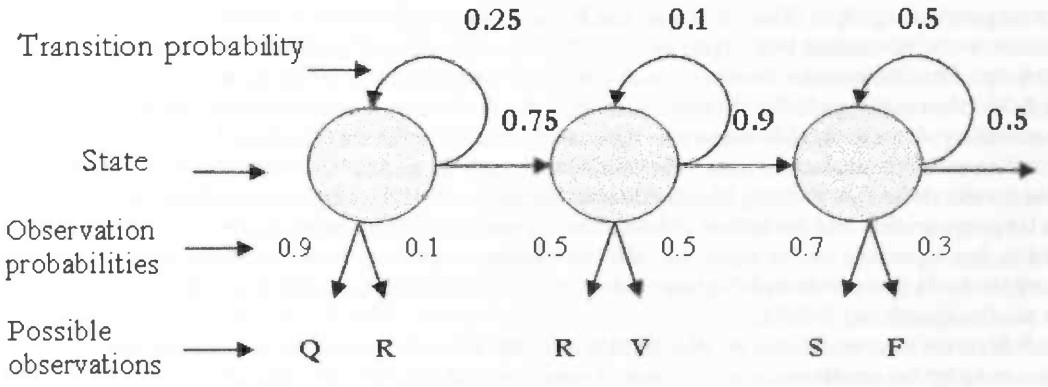


Figure 3: Simplified example of an HMM. There are several possible observations per state, each with a certain probability. There are also several transitions possible from a state, each with its own probability.

2.3.1.3.1 Small- vocabulary whole-word and part-of-word HMMs

a) Whole-word models

Vogler & Metaxas (1997) tried several techniques to adapt HMMs to sign language recognition. They used American sign language in their experiments. They experimented with data collected with three orthogonally positioned cameras as well as with data from magnetic trackers. Both worked, but since the computer vision method took more time to calculate positions and orientations, they used the trackers for the major part of the data collection. They used only position and orientation of the hands as features. In this experiment, no data gloves were used, only magnetic trackers.

The authors' main concern in this project was how to deal with co-articulation effects in continuous signed speech. Co-articulation affects signs just as it does sounds. Furthermore, if two signs made in succession are made in two different locations (e.g. at the head and in front of the chest), an extra movement takes place as the hands move from the first location to the second. These extra gestures which have no meaning are called *movement epenthesis*. See section 1.1.5 for more information.

In speech recognition, co-articulation problems are handled by modelling bi- and tri-phone context dependent HMMs. That is, models are made not for isolated sounds, but for every possible two- or three-way combination of sounds including the co-articulation effects that occur between them. The authors tried this approach for their whole-word sign models: they made bi-sign models. Because the epenthesis movements can vary quite a bit in their length, the HMMs representing sign combinations had to be large (i.e. contain many states). If they had been small, long epenthesis movements could not have been modelled successfully. So large models were created, but with extra skips, so that the HMM could also model two signs with a short epenthesis movement between them.

This approach was not considered viable, though, because when adding a new word to the vocabulary, new HMMs for every possible combination with all the others have to be made and trained. The size of the model bank grow explosively with vocabulary size and the amount of training data needed would be tremendous. That is why the authors explored an alternative: modelling epenthesis movement explicitly, that is, making a separate HMM for every possible epenthesis movement and recognising such a movement simply as 'epenthesis'. Since there is only a limited amount of places to start and end a sign, there is only a limited amount of transitions between these points. So there is only a limited set of possible epenthesis movements. These can be modelled and used to detect epenthesis in continuous signing.

Finally, the authors looked at bigram models for sign language also. Bigram models are language models that give a measure of probability for word combinations: certain words are more likely to occur after one

another than others (e.g. a combination of 'go' and 'to' has a higher probability than 'go' and 'moo'). These probabilities are calculated from the corpus of a language. The information is used for example in speech recognition to help decide which models are the most probable. Vogler & Metaxas wanted to do the same for sign language recognition. They could not use statistical information calculated from real sign language use, because such information is not (yet) available for sign languages. The authors did however collect statistical data from their own – limited – vocabulary and used this to create bigram models.
Vogler & Metaxas experimented with combinations of the techniques described above. They used a 53-word vocabulary, from which 486 sentences were created with lengths varying from 2 to 12 words. They used word accuracy to measure success. They found that modelling epenthesis movements explicitly provided better results than creating bi-sign context dependent HMMs, even when the latter used the bigram language models and the former did not. The bigram models improved the results for both approaches, but especially for the explicit modelling technique. The best results were a word accuracy of 96% for epenthesis movement with bigram models, and 92% for bi-sign context dependent HMMs with bigram models (perplexity 0.019).
Vogler & Metaxas also tried using parallel HMMs to model the parallel processes of the left and right hand in a sign, merging the results at the word limits. Again, they did not use instrumented gloves, only magnetic trackers. With the parallel method, too, they achieved good results of around 94% on a 22-word vocabulary (a perplexity of 0.045). However, they soon realised that the problem with building whole-word HMMs is scalability to large vocabularies: when a separate model has to be built for each word in the vocabulary, the bank of models soon becomes huge and the amount of training data grows rapidly also. This is why Vogler & Metaxas abandoned the idea of a whole-word model, and started modelling parts of signs instead. For that reason, I will only briefly evaluate the methods described here, and move on to their more recent experiments with HMMs representing parts of signs.

The authors themselves already pointed out the greatest difficulty of the whole-word HMM approach: scalability. Adding to the vocabulary is not easy, since a new model has to be trained for each sign, which takes a lot of training data. And searching through large banks of models can become a problem in itself, although Chen et al. propose some solutions to this difficulty (see section 2.3.1.3.2). Apart from this, the authors collect and use only information on position and orientation of the hand, which is hardly exhaustive. The system is expandable, in that data from other sources could be added to the feature set, and all features could then be used to train the HMMs. But the small amount of relevant features used and the smallness of the vocabularies make it difficult to say how well this method could handle signs that differ only slightly. Ability to generalise over different signers also remains uncertain. However, it is more interesting to look at the authors' more recent experiments.

b) Part-of-word models

Automatic sign recognition researchers using HMMs soon realized that modelling whole words did not scale well to large vocabularies. Because an HMM needs a lot of data to train on, creating models for every word in a large vocabulary would require a huge amount of training examples and result in a large bank of models that had to be searched through. Rather than creating and training a new model for each new sign, it would be better to make a limited amount of models for a limited amount of sign parts. If a set of basic sign parts could be found from which all signs could be built, models for those parts could be used to recognise every sign. In speech recognition, phonemes are modelled for this purpose. Sign recognition researchers tried to find a way to do the same for sign languages. There is a difference in this respect between sign and speech, however. Speech phonemes always occur sequentially. This is obvious, since you only have one vocal tract and can only make one sound at a time, though it could be argued that prosodic markers are also phonemes, and these can occur simultaneously with other speech phonemes. In sign language, at any rate, phonemes (parts of signs which are essential to its meaning, such as handshape and location) certainly occur simultaneously as well as sequentially. This is a difficulty researchers have to solve if they are to models sign parts.

Vogler & Metaxas (1999a) tried to find a set of sign language phonemes that is always sequential. That is why they used the sign phoneme model of Liddell & Johnson, which segments signs into Movements and Holds. Since there are no existing sources of MH-descriptions of signs, the authors had to make those descriptions themselves. Each M- or H-segment has a bundle of articulatory features attached to it. In the

original theory of Liddell & Johnson, these articulatory features are hand configuration, orientation, location and motion (the last one not for Holds, of course). But Vogler & Metaxas used only location for the Hold phonemes and a motion description for the Movement phonemes in their project.
The authors experimented with American sign language. They used instrumented gloves and magnetic trackers to collect data. They claim, though, that data collection could be done just as easily with a video camera. The features they used were divided into local features – these were position and velocity of the right hand – and global features – which line or plane does the movement fit? The problem of movement epenthesis was solved by modelling each epenthesis movement as a separate phoneme, though ultimately the authors would like to simply recognise all these movements with one general model for 'epenthesis'. Thus, instead of building models for entire signs, the authors modelled a number of possible Movement- and Hold phonemes. Each sign consists of a sequence of movements and holds, and each movement- and hold segment has a certain bundle of features attached to it. When it is known that a sign consists of a Hold, a Movement and a Hold (HMH), the best combination of an H-model, a M-model and an H-model that provides the observed features is sought. The observed features are the position of the right hand and motion descriptors. The combination of three models that is found can then be looked up in a dictionary to see which sign this combination represents (just as, in speech recognition, a combination of phonemes represents a certain word).

The authors experimented with a 22-sign vocabulary. They created 499 sentences containing 1610 signs samples overall. 99 sentences were used for testing, the rest were training material. The authors only used features from the right hand. Using word-accuracy as an error measurement they obtained an accuracy of 91% using global features, and 88% using local features, which is comparable to the accuracy they achieved with whole-word HMMs. The difference is, of course, that with phoneme HMMs, new words can be added to the vocabulary without having to train extra models: it is just a matter of adding the appropriate description to the dictionary. This will have to be done by hand, since no MH-descriptions of signs exist (yet).

The new approach of modelling phonemes instead of whole signs solves the problems of adding signs to the vocabulary and handling large vocabularies, mentioned in the part a) of this section. Accuracy is also good at around 90%, though for a vocabulary of only 22 signs (perplexity 0.045). And with the modelling of epenthesis movements, the system seems capable of handling continuous signed speech. It also seems to generalise well over intra-signer variability.
However, I assume the system was only trained and tested on one signer (this is not mentioned in the authors' paper), so how well it can generalise over different signers is not certain. The authors also do not use handshape or non-manual features, and use orientation only as a motion descriptor (wrist roll), not as a feature for a hold-phoneme. It is probably the small size of the vocabulary that allows the system to achieve such good results on the basis of location and motion data only. It is difficult to add the missing features without the aid of a glove or a camera, but with these, expanding the feature bundles associated with the M- and H-phonemes should be possible. However, I wonder how well this would work: with more features, more possible combinations arise and more different M- and H-phonemes will be the result. How will the authors deal with this? If handshape changes during a motion, will this be regarded as a sequence of different M-phonemes (in each of which the feature 'handshape' is slightly different), or will the handshape feature simple not be associated with movements, only with holds? These are questions that make it difficult to determine how well expandable this system is. It is also not clear whether this system works in real-time.
And then there is the matter of applicability to different sign languages. If all sign language signs everywhere consist of a certain set of M- and H-phonemes, then this system can be used for all sign languages once the entire set of phonemes has been learnt and modelled, which would be a great advantage. But to what extent this is the case, is not certain, because there is not much information available on this subject. The basic technique of modelling movements and holds and associating features with them can be applied to every sign language, though. Finally, the signer needs to wear hardware on his body, which is not preferable. But using location and motion features is cognitively plausible, even if they are collected with the aid of magnetic trackers.

2.3.1.3.2 Large-vocabulary whole-word HMMs

Chen, Gao, Fang, Yang & Wang (2003) have worked on the recognition of Chinese Sign Language for several years (Chen et al., 2003; Fang et al., 2003, 2002; Wang et al., 2002; Gao et al., 2000). Their aim is to build a system that can deal with realistic, large vocabularies. The authors train HMMs to represent whole words. When a large vocabulary is used, this means that a large bank of models has to be created – one model for each sign in the vocabulary. Searching through such a large model bank takes a lot of time and resources. To deal with this problem, Chen et al. have developed several techniques to reduce the computational load of searching through large dictionaries of whole-word HMMs. These include:

- varying the number of states in each HMM, so that a model is no larger than it has to be;
- clustering the vectors associated with HMM states for easier probability computing;
- evaluating only HMMs with states above a certain threshold;
- organizing the HMMs in a hierarchical decision tree, in which those features that are most differentiating are evaluated first;

All these techniques reduce time needed and computational load. The fewer states an HMM has, the less time it takes to calculate its probability of producing a certain output, so creating only as many states as necessary per model is useful. Clustering output vectors makes it easier to calculate probability, which also saves time. Discarding HMMs of which all states are below a certain activity threshold, meaning that they have a very slim chance of producing any part of the observation sequence, reduces computation, too. And finally, by organizing their models in an hierarchical decision tree, the authors reduce the amount of models that has to be searched through quickly: for instance, at the top of the tree they put the feature "one-handed or two-handed". This feature can be determined quickly from the data and when it is resolved, a large part of the models can immediately be discarded.

In 2003, the authors presented a Chinese sign language dialogue system which could recognise continuous sign language and synthesise sign language from speech as well. The authors used instrumented gloves on each hand, a magnetic tracker on each hand and a reference point for the trackers worn on the waist or the back. They used a 5113-word vocabulary. Six signers each performed every sign twice, resulting in a database of 61356 samples. Recognition took place in real-time. The accuracy on word recognition was 91.6% when tested on signers the system had trained on. This drops to 83.7% when tested on unseen signers. In an earlier project, the system was trained and tested on a single signer and achieved a word accuracy of 94% (Wang et al., 2002).
The vocabulary words were used to create 750 sign sentences. Two samples of each sentence resulted in a database of 1500 sentences. On sentences, recognition rate was 90.8% (perplexity 0.0007). The authors dealt with the problem of movement epenthesis by modelling the epenthesis movements separately and recognising those movements as 'epenthesis'. I assume that they deal with finding word boundaries in continuous signed speech by using some variant of the Viterbi algorithm, but this is not clear from the papers of Chen et al. How the authors deal with co-articulation effects is also unclear.

The method of Chen et al. achieves good recognition results for six different signers, on single words as well as on sentences. How long the sentences are, and whether they have a constricted grammar or not is not clear, though. In any case, the authors do not describe methods of dealing with context-dependent signs, so I assume they did not use any in their sentences.
The system uses all relevant features of signs, except non-manual features. It can clearly deal with large vocabularies in real-time and in different environments (for a glove-based system, this is not a problem). It also seems capable of handling continuous signed speech, though how exactly this is achieved is described only very concisely.
But there are certain problem areas as well. The first is the expandability of the vocabulary. The authors create one HMM for each sign, and an HMM needs a lot of data to train well. Furthermore, the more different signers a system has to be able to understand, the broader the training data need to be, and therefore, the larger the training set has to be. The amount of training that goes into one models makes it difficult to add signs to the vocabulary. And even if you include the entire vocabulary in one go, there is still the problem of new signs emerging. Language is not static, so new signs will have to be added from time to time, and the nature of HMMs makes this cumbersome.

Then there is the problem of signer-independency. The high recognition percentage for signs from one signer (94%, perplexity 0.0002) suggests that the system can learn to generalise quite well over intra-signer variations. But when the system has to deal with data from six different signers, accuracy drops to 92% (same perplexity). Will it drop more when it is tested with 10, 20, 50 signers? Or can this only be avoided by adding more and more training examples per sign? Neither alternative is appealing. The drop in recognition rate when tested on unseen signers (from 92% to 84%) also suggests that this system does not generalise well over different signers.
Of course, a system can still offer good perspectives even when it is not signer-independent. But in that case, it must be possible to train the system on *any one* person. That is, a user must be able to train or fine tune the system to his own particular signing characteristics, the way it is nowadays often done for speech recognition systems. This is a possibility, but I don't see how it could be done with the system of Chen et al. Their sign models are created from the training data, and since there is one model for each word, the amount of training data is huge. A user cannot be required to perform 5113 signs four times or so in order to get his system to work. Apart from the huge amount of work, the creation of models is something that must be done accurately in a lab, it cannot be done by an inexperienced user at home. And it is not clear how one could fine tune whole-word HMMs that are already made. In speech recognition it is different: there, HMMs represent phonemes, of which there is only a limited number. These can easily be fine-tuned by the user.
Finally, the system is not very expandable. It is not easy to see how it could be made more signer-independent, for instance, or how non-manual information could be added. And of course it has the same problems all glove-based methods share: the need for hardware and cognitive implausibility.

2.3.2 Image-based methods

2.3.2.1 Fuzzy logic expert system

Fuzzy logic is an approach in which the boundaries between values of parameters are taken to be soft, not hard. That is, a position is not either A or B, and a value of a measurement need not be either 1 or 2. Instead, values are 'mostly' A and 'slightly' B, values can overlap. For a more extensive introduction into this subject, I refer to Ross (1995). An expert system is a system that can make decision based on the rules in its rule database. For more information, see for example Russell & Norvig (1995).

Holden, Owens & Roy (1999) used fuzzy logic in the process of sign language recognition. They used signs from Australian sign language (Auslan) and collected their data by means of a camera. The signer had to wear a colour-coded glove, and make an artificial handshape at the start and end of each sign. A hand tracker extracted certain values from the video images to form a 3D model of the hand. This model contained values for a number of finger joints. For each joint in the model, the angle of bend (joint flex) was determined. For a couple of joints, amount of rotation was also determined. With this model, the system possessed a set of values that provided information about handshape and hand rotation.
The joint flex values were then used in a rule-based, fuzzy system. Each joint could have certain values: for example: the middle joints of the fingers could have the values *straight, slightly flexed* and *flexed*. These were fuzzy values; a certain amount of flex (say 15 degrees) would be 80% *straight* and 25% *slightly flexed*. The advantage of this approach is, that it can deal with uncertain data more easily. For a digit flex value of 15 degrees, a hypothetic sign involving a *slightly flexed* digit is not immediately discarded. It does, however, have a smaller likelihood than another hypothetic sign in which this digit is *straight*.

Holden et al. gave every joint values and percentages in this way. They also determined motion, not motion of the entire hand, but motion of the fingers, such as finger wiggles. This was done by determining the number of directional changes in the flex variables over time. For example, if the value of flex of a certain digit is first 15%, then over time becomes 20%, then drops to 10% and then goes up again to 13%, then the number of directional changes, the number of 'uphills' and 'downhills', is 3. These numbers were used to determine which sort of motion (if any) took place in a sign.
With all values and percentages determined, the data could be used by rules. Holden et al. created rules by hand to determine postures and signs. Postures were determined by the digit flex values. The rules were something like: IF flex_of_middle_joint_of_finger_1 IS *flexed* AND flex_of_middle_joint_of_finger_2 IS *flexed* AND etc.... THEN posture IS *handshape1*. All rules were given an activation level. The activation

level of a rule was the minimum of activation levels of its components. Thus, if the likelihood of the flex of the middle joint of finger_1 being *flexed* was the lowest, say 20%, then the activation level of the entire rule was also 20%. The rule with the highest activation level was the best fitting rule, and its conclusion was accepted. As an alternative, the best *n* rules could be returned, with their associated probabilities.
Postures were thus determined with these rules. The actual sign was then identified as follows: postures were determined for the start and end of the sign sequence. Then rules using start posture, end posture and motion values for the time in between, were used to determine the sign. These rules, too, were made by hand. Thus, a sign classification rule would look like this: IF posture_start IS *handshape1* AND motion_of_middle_joint_of_finger_1 IS *slightWiggle* AND motion_of_middle_joint_of_finger_2 IS *bigWiggle* AND etc... AND posture_end IS *handshape4* THEN sign IS *'spoon'*.
Again, the activation level of such a rule is the activation of its least activated component, and the conclusion of the most active rule was the sign recognised. The system was tested with a 22 word vocabulary, and in the end it could recognise 21 of these.

With a recognition rate of 95% (perplexity 0.045), this system is certainly successful, be it only for a small vocabulary. But it has severe limitations. The most important one is the fact that only the features shape and orientation of the hand can be obtained and used. The system does not allow for motion of the hand, location or non-manual features to be detected, because it only processes the gloved hand, not the entire body configuration. It is therefore not easy to add modules to extract these data. Furthermore, all classification rules are made by hand, a task that becomes impossible if more features are used and the vocabulary is enlarged. The system only recognises single words, and even these have to be indicated with a special posture at the start and finish. Adding to the vocabulary is not easy, since new rules have to be made by hand for the new sign. It is uncertain whether the system works in real-time, and whether it could do this with a vocabulary of several thousand signs. Finally, the nature of the system is such that it is not easily expanded to deal with these problems: the authors do not propose a method for generating rules automatically, and without such a method, this system cannot deal with real-world-sized vocabularies. So in conclusion, it is an advantage of fuzzy logic that it can deal easily with the vagueness of the boundaries between different handshapes or orientations, but the limitations of this fuzzy system are so severe, that I do not think it is suitable for dealing with natural sign language.

2.3.2.2 Hidden Markov Models

For an explanation of hidden Markov models (HMMs), see section 2.3.1.3.

2.3.2.2.1 Whole-word HMMs

Starner, Weaver & Pentland (1998) used HMMs for sign language recognition. They experimented with American sign language (ASL). They collected their data with cameras and built one HMM for every sign in the vocabulary. They experimented with different types of data collection: a desk-mounted camera and a cap-mounted camera. In the first setup, a camera was mounted on the desktop, so that it was in front of the signer, like a normal listener would be. In the second setup, a small camera was worn in a baseball cap, from where it looked straight down onto the signer's hands. The hands were tracked using skin colour, so no gloves were needed.
The authors extracted a feature vector with information on position, velocity, orientation and movement of the hands. With these features as data, they trained whole-word HMMs for a 40-word vocabulary. They created about 500 5-word sentences with signs from the vocabulary and used four fifths of them for training, and the rest for testing. All data was collected from a single signer.
Starner et al. performed several experiments. They looked at the results of using a restricted grammar, demanding that all sentences be in the form: "personal pronoun, verb, noun, adjective, (same) personal pronoun", versus an unrestricted grammar. They demanded that the signer remained in the same place during signing, because they used a feature 'absolute position on the screen'. But because this is not a very realistic approach, they also experimented with discarding this feature. Finally, they experimented with local vs. global features, that is, features that change from one moment to the next vs. features that stay the same for the entire sign.
The word accuracy results were high for the cap-mounted system (97% with unrestricted grammar, 98% otherwise), but lower for the desk-mounted system (75% with unrestricted grammar, 92% otherwise). The

perplexity in this experiment was 0.025. Discarding absolute position as a feature actually worsened accuracy, from 92% to 87% for the desk-mounted system, but the authors still felt that it proved the system's capabilities of dealing with a moving signer. They concluded that HMMs offer perspectives for the future, that a cap-mounted viewpoint improves recognition greatly and that using global features, that is, features such as bounding ellipse of a sign, or axis of least inertia, provides better recognition than using local features, such as position and velocity.

Evaluating this early method with my criteria, some problems become apparent. Let us first look at the cap-mounted system. This system has a very good recognition rate, but two problems are inherent to such a system: it cannot have access to non-manual information, because the face is not visible from such a camera position. And secondly, the system is cognitively implausible, because it has a view not normally seen by a listener. A signer signs to convey information to someone opposite him, not straight above his hands. Thus, some sign features that are significant could be invisible from straight above, such as the shape of the hand held below the other had. This is perfectly visible to a listener positioned opposite the signer, but not to a camera located straight above. And conversely, the system might read significance in features that cannot be seen by a listener en can therefore never be significant, such as the position of fingers obscured from the listener. This is why cognitive plausibility is important. The desk-mounted system does not have these problems, but its accuracy is a lot lower than that of the cap-mounted setup. The system does use almost all important features, only non-manual features are ignored. Its high recognition results for a single signer when grammar is restricted suggests that the system can generalise well over intra-signer differences. It works in real-time, too. The desk-mounted system should in principle be expandable to use non-manual features, too: once a face recognition device has been added, the features resulting from it could be used along with the others to train HMMs. However, the cap-mounted system does not have this possibility, because it has not view of the signer's face. How well the system would work against an interfering background is not clear from Starner et al.'s paper.
These are the criteria the system meets more or less. Other criteria present a problem. Adding to the vocabulary is not easy, since a new HMM has to be trained for each new sign, and an HMM needs a lot of training examples. How well the system could generalise over different signers is not clear, but results from other whole-word HMM methods (Vogler & Metaxas, 1997; Chen et al., 2003) are cause for pessimism on this account (see section 2.3.1.3). And its applicability to other sign languages is a problem, as it is for all whole-word HMM systems, because all the training has to be done again for each new language. Handling large vocabularies is also difficult for whole-word modelling systems because of the large amount of models, though Chen et al. propose solutions to deal with these problems. And finally, continuous signed speech is a problem. Though the system performs well on recognising sentences, this accuracy drops to 75% when a restricted grammar is not used. The results are higher for the cap-mounted system, but this has so many other problems, that it is probably not be feasible for real-world recognition. Apart from the accuracy, though, there is the problem of finding word boundaries and movement epenthesis in continuous sign language. The authors do not mention how they propose to deal with these matters. I assume that their sentences are signs put in a row, not true sentences with co-articulation etcetera, though I am not certain from their paper. So recognition of real continuous sign language is still an issue for this system.

2.3.2.2.2 Part-of-word HMMs

For an explanation on using HMMs to model parts of words, see section 2.3.1.3.1.b.

Bauer & Hienz (2001) experimented with German sign language. They tried recognising sign language with HMMs, like many other researchers did. Like Vogler & Metaxas (1999), they recognised the advantages of modelling parts of signs instead of whole signs. They used a different approach, however. Where Vogler & Metaxas used Liddell & Johnson's phonemes as the parts-of-signs to be modelled (see section 2.3.1.3.1.b), Bauer & Hienz proposed modelling so-called subunits. These need not have a linguistic justification as parts of signs. They are simply clearly distinguishable parts in which most signs can apparently be divided. The subunits were extracted from the data using k-means clustering. In a dictionary, it was recorded which sequence of subunits formed which sign. Bauer & Hienz then created and trained HMMs to represent these subunits. Consequently, they recognised an unknown sign by finding the

concatenation of subunit models that had the greatest probability of producing the observed data. This combination of subunits could then be looked up in the dictionary to find out which sign they formed. The advantages are the same as for other part-of-word methods: instead of modelling all signs, only a limited number of sub-sign units needs to be modelled, with which all signs can then be recognised. This approach is more practical than that of Vogler & Metaxas (1999), though, because Bauer & Hienz do not have to develop the descriptions of signs in terms of subunits by hand: these are generated automatically in the training phase.
The authors used a colour camera to capture image data. The signer wore coloured gloves. The glove of the dominant hand had separate colours for each finger and for the front and back of the palm. This helped to determine handshape and orientation. The position of the shoulders and the central body axis was determined to form a body-centred co-ordinate system. The features that were extracted from the image data, were: co-ordinates of the dominant hand and distance between hands, distances of all coloured markers of the dominant hand to each other (this is a way of distinguishing between handshapes), size of the coloured areas (for handshape and orientation) and angles of the fingers with the x-axis (for orientation). So all relevant features of a sign, except non-manual features, were represented in some measure.
The creation of part-of-sign models then went as follows: for each frame of data, the feature vector was extracted. The vectors were then clustered, using k-means clustering. The average vectors of the resultant clusters were used as subunits. A translation of signs into these subunits was made simply by determining and recording in which cluster each feature vector of a sign had ended up. So if a sign consisted of three vectors, of which the first ended up in cluster alfa, the second in cluster beta and the third in cluster alfa again, then the translation of that sign into subunits would be: alfa-beta-alfa. These translations can easily be generated automatically.
After the subunits were determined, HMMs could be trained for them. Recognition of a new sign was then done in terms of which sequence of subunits fitted the data best. This sequence could be looked up in the dictionary to find the sign which it represents, which is the sign recognised.
Bauer & Hienz tested their approach with a 12-word vocabulary. The training- and test corpus each consisted of one instance of each sign, so they each contained only 12 signs. Clustering resulted in 10 different subunits. The recognition rate was 81%. They felt that the corpus of words was too small to find good subunits: in speech recognition, about 200 subunits are usually used. The authors feel that with a larger corpus of signs, better subunits could be formed and better results obtained.

This method was performed with such a small vocabulary, that it is difficult to examine how well it fits the criteria specified earlier. The recognition rate is not too good: 81% for a the 12-word vocabulary (perplexity 0.083). A small vocabulary is beneficial to the recognition rate, because the distinctions made need not be very fine. On the other hand, the authors suspect that with more signs, better subunits could be created, which would aid the recognition process. So whether results will increase or drop with a larger vocabulary is not easy to estimate.
The system does use almost all significant sign features, only non-manual features are ignored. And it is expandable in that it could add information from a face tracker to the feature vector, and cluster these enlarged vectors. Being a system modelling subunits, it should have no problem with adding signs to the vocabulary: this is simply a matter of determining in which clusters the sign's feature vectors would fall, and adding the subunit description to the dictionary. Handling large vocabularies should also be no problem: searching through a large dictionary is a lot easier than searching through a large model bank, because a dictionary can be ordered. And the model bank will remain small, of course, because only subunits are modelled, and there are not many of these, compared to words.
But nothing can be said about the system's ability to generalise over different signs of the same user and over different users. It is not even clear from the paper whether training- and test corpus were different. This is a shame, because it is very interesting how general these clustered subunits would be: is a clustering obtained from data from one or more signers useful in analysing other unseen signers? Or are they very much tailored to the data set they were trained on? And how about different sign languages: can subunits extracted from one language be used successfully to express signs from a different language? These questions remain unanswered for now. It is also unclear whether this system could work in real-time, although it would appear so, because computational demands are not large. About continuous signed speech recognition it is also difficult to draw conclusions. The authors claim that it would be possible and propose some methods of finding the best sub-unit sequence for a large observation sequence, but they do

not provide details or test results. Finally, it is unclear how well their visual tracking system would work against a cluttered background. The use of gloves aids the tracking process, but what if there are competing areas of colour in the image? Self-organising subunits are promising, but more research needs to be done to determine exactly how useful they could be in automatic sign language recognition.

2.3.2.3 Whole-word Markov chains

Markov chains are models that resemble hidden Markov models (HMMs) somewhat (for information on HMMs, see section 2.3.1.3). Markov chains, too, consist of states, observations and transitions. But whereas HMMs have several possible observarions per state, with varying probabilities, Markov chains only have one possible observarion per state. The state more or less *is* the observarion, since you always get that observarion when you are in that state. For the rest, Markov chains operate in the same way as HMMs. At each time-step, the model is in a certain state. It generates the observarion that state contains. In the next time-step, it moves to the next state (this could be the same state again, if there is a transition from the state back to itself) via one of its possible transitions. Each transition has a certain probability of happening. In the next state, the observation of that state is generated, then the model moves again to a next state, etcetera. This means that a Markov chain model can generate a certain output sequence with a certain probability. An example is shown in fig. 4: the probability of this model generating the sequence 'QQQRS' is 0.25 * 0.25 * 0.75 * 0.9 = 0.04, or 4%.

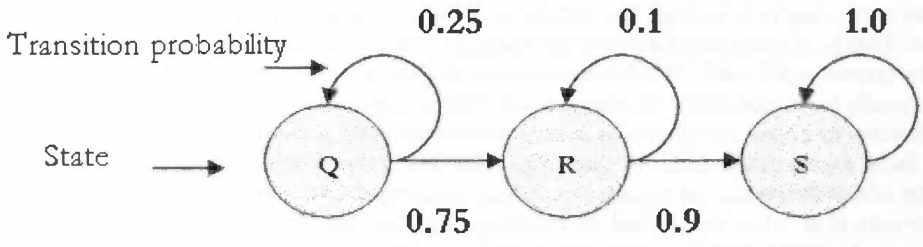


Figure 4: Example of a Markov chain model. This model has three states and transition probabilities between some of the states. There is only one possible observation per state, so 'state' and 'observarion' are used as equivalents.

Bowden, Winridge, Kadir, Zisserman & Brady (2004) used Markov chains to model whole signs from British sign language (BSL). The advantage of Markov chains over HMMs is that, unlike HMMs, Markov chains do not need many training examples. Bowden et al. build their models from as little as a single training example.

Bowden et al. collected their data with a single camera. The hands were tracked using skin colour, so no gloves needed to be worn. The authors experimented with British sign language. They extracted the features hand arrangement (configuration of the hands), hand position, handshape and motion from the data. Hand position, handshape and motion were only extracted for the dominant hand. The authors obtained their features in the following way: first, head an shoulder contour were found. This provided a body-centred co-ordinate system in which the most important body locations can be estimated. Then the hands were tracked. For each movie frame, the hand position, motion, shape and location were found and analysed in terms of categories. These categories are given in Appendix I. So if hand position in a certain frame was found to be (241,304), the system checked in which category this position would fall. Suppose everything from (200,200) to (400,400) is considered 'right side of face', then this position is recorded as 'right side of face'. The same was done for motion and hand arrangement. The three features were translated into a binary format and concatenated to form one long binary vector. One such vector was created for each frame in a sign movie.

With these feature vectors, the authors build models. They first create a lookup table, in which each different vector is associated with a unique symbol. These symbols are used as the output of the Markov chain states. Models are created on the basis of one training example. They are built by adding a new state for each new symbol encountered in the example sign. I assume that transitions are created according to their relative frequency in the sign. For example: in the vector sequence alfa-beta-alfa-beta-alfa-gamma, after a vector alfa a vector beta is found twice, and a vector gamma once. The transition from state alfa to state beta will then be 0.67, and from alfa to gamma 0.33. But this is not described.
In this manner, a model was made for each sign in the vocabulary. Recognition was then performed as follows: for an unknown sign, the binary feature vectors were extracted for all frames. This sequence of vectors was translated into a sequence of symbols by appointing to each vector the symbol of the vector in the lookup table that it resembled most closely. Then it was calculated which model had the greatest probability of producing such a symbol sequence. The sign that model represented was the sign recognised. The signs were offered sequentially to the system. It used the Viterbi algorithm to find the best sequence of signs (models) to fit the observed data.
The authors used a 49-word vocabulary of randomly selected signs. They collected their data from a single signer. 249 instances were recorded, averaging 5 instances per sign. One instance per sign was used for training, the rest formed the test set. The recognition rate was 84% (perplexity 0.020). This rose to 98% (perplexity 0.023) when six signs were removed that could only be distinguished by their non-manual component, a feature which Bowden et al.'s system does not use.

The recognition rate of this system is very good. When only sign are used that can be distinguished with the features that are available to the system, it even reaches 98% (for a perplexity of 0.023). This suggests that perhaps the system could reach such a high percentage for its entire vocabulary if modules for extra features are added. That is, if recognition modules for hand orientation and non-manual component are added, perhaps the system could reach 98% for its entire vocabulary. However, the vocabulary is not very large, and this is usually beneficial to the recognition rate. The system uses all relevant sign features, except for non-manual features and hand orientation. It is very well expandable: a face tracker could easily be added to analyse facial expression in terms of categories. The results this provides could be added to the binary vector, after which the rest of the system can remain unchanged. That this system is well expandable was proven by Bowden et al. when they added the handshape analyser between 2003 (Bowden et al., 2003) and 2004 (Bowden et al., 2004). Additions to the vocabulary are not a problem for this system, even though it uses whole-word models, because the Markov chain models can be built so easily. One example is enough. Dealing with large vocabularies does remain a problem, though: searching through a large bank of models takes time and resources. But Chen et al. propose some techniques to deal with this problem (see section 2.3.1.3.2). The system works in real-time, and can work against cluttered backgrounds, as demonstrated in Ong & Bowden (2004). It is cognitively plausible, because it uses the same viewpoint as a human being, that is, it uses only information that is available to humans listeners. And finally, the signer does not have to wear hardware on his body.
It is uncertain how well the system generalises. The high recognition percentage, especially after removing indistinguishable signs, suggests that the system generalises well over intra-signer variations. But how well it would work on different signers is not clear, since it has only been trained and tested on one signer. However, the feature analysis in terms of categories provides a generalising effect early in the recognition process, which could well be capable of generalising over different signers. It is this, too, which makes the system cognitively plausible: features are analysed in terms of 'significant categories': no human sees the difference between two centimetres more to the left or not, so that is not significant. Significant are only differences in categories: right side of face, in front of face, or left side of face; right side of chin, in front of chin or left side of chin, etc. How many categories should be created is a manner for sign linguistics research. Bowden et al. do not describe a motivation for their categories, so I do not know how linguistically correct they are.
How well Bowden et al.'s system could deal with continuous signed speech is also uncertain. They do offer their data in one stream, but I assume these are single signs concatenated, not actual sentences with co-articulation effects and movement epenthesis. In any case, the authors do not describe methods to deal with these problems.
And finally, there is the issue of using this system for other sign languages. Because whole-word models are used, new models have to be made for each new language, of course. But this is not a problem, since model creation is easy and fast. The feature extraction system could be applied to other sign languages

without problems. However, it is important to consider if the same feature categories are suitable for all sign languages. Are all sign languages alike in how much detail they distinguish? Or are there categories in some languages (position next to the temple, next to the cheek) which are not distinct in others (where they would both simply be 'next to the head')? This is unclear, because hardly any comparative linguistic research has been done for sign languages.

2.4 Conclusions

In the previous section, I reviewed several methods of automatic sign language recognition. In this section, I will compare them all and choose the best method, which I will then apply to the recognition of Dutch sign language. I have given an overview of the findings described in the previous section in table 1. It lists all methods, and all criteria that were used to evaluate them. A plus sign means a method meets a criterion, a minus sign means that it does not, and a question mark means that it is unclear whether it does or not, either because this information was not provided by the researchers, or because the method was too small-scaled to allow for a conclusion on this criterion. Recognition accuracy is given simply as a percentage. The amount of sign features used is given as a number between 0 and 5 (for the five features: handshape, location, orientation, motion, non-manual component). For convenience, the (first) author names and paragraph numbers are also provided.

Table 1: Overview of criteria for each method discussed. The number of sign aspects used lies between 1 and 5, since there are 5 aspects to each sign (see section 1.1.2). A plus sign means a method meets a criterion, a minus sign means it does not, and a question mark means that it is unclear. +? means 'probably', -? means 'probably not'.

	GLOVE-BASED				IMAGE-BASED			
Method	Machine learning	Artificial neural networks	Part-of-word HMMs	Whole-word HMMs	Fuzzy expert system	Whole-word HMMs	Part-of-word HMMs	Whole-word Markov chains
Author	Kadous	Vamplew	Vogler	Chen	Holden	Starner	Bauer	Bowden
Section	2.3.1.1	2.3.1.2	2.3.1.3.1.b	2.3.1.3.2	2.3.2.1	2.3.2.2.1	2.3.2.2.2	2.3.2.3
Size of vocabulary / perplexity	95 / 0.011	52 / 0.019	22 / 0.045	5113 / 0.0002	22 / 0.045	40 / 0.025	12 / 0.083	49 / 0.020
Recognition accuracy: (known signer(s))	80%	94%	91%	92%	95%	75%	81%	84%
Recognition accuracy: (unknown signers)		85%		84%				
Is in principle capable of collecting all sign aspects?	-	-	-	-	-	+	+	+
Number of sign aspects actually used:	4	4	2	4	2	4	4	3
Expandable to use more sign aspects?	+	+	?	-	-	+	+	+
Can add easily to vocabulary?	+	+	+	-	-	-	+	+
Can handle large vocabularies?	?	+	+	+	-	-	+	+/- (would need extra algor.)
Intra-signer generalisation?	+	+	+	+	+	+	+	+
Inter-signer generalisation?	-	-	?	-	?	?	?	?
Applicable to different sign languages?	+	?	?	-	+	-	?	+
Can handle continuous signed speech?	?	-	+	+?	-	-?	+?	?

Table 1 (continued)

	GLOVE-BASED				IMAGE-BASED			
Method	Machine learning	Artificial neural networks	Part-of-word HMMs	Whole-word HMMs	Fuzzy expert system	Whole-word HMMs	Part-of-word HMMs	Whole-word Markov chains
Author	Kadous	Vamplew	Vogler	Chen	Holden	Starner	Bauer	Bowden
Section	2.3.1.1	2.3.1.2	2.3.1.3.1.b	2.3.1.3.2	2.3.2.1	2.3.2.2.1	2.3.2.2.2	2.3.2.3
Works in real-time?	+	?	?	+	?	+	+?	+
Works against cluttered backgrounds?	+	+	+	+	+	?	?	+
Works without wearing hardware?	-	-	-	-	-	+	+/ (coloured gloves needed)	+
Cognitively plausible?	-	-	+	-	-	+	+	+

The criterion of applicability to different sign languages was especially important, of course, because the aim of my project is to achieve automatic recognition for *Dutch* sign language. So methods that scored a '-' on this point were automatically unfit for my purposes.
From this result table, it can be seen that the whole-word Markov chain system of Bowden et al. (2004) is very good. Though it only uses three features at the moment (orientation and non-manual component are not used), it has proven to be easily expandable to include new features. Furthermore, it is very robust in that it needs no hardware on the signer's body and can work against cluttered backgrounds. It can create models easily and quickly, and seems to generalise well over intra-signer variations (about inter-signer variations, no information is available). And although the recognition rate is 84%, which is low compared to some of the other methods, it rose to 98% when signs indistinguishable to the system (because they only differed in facial expression, for example) were removed. This suggests that the system could be extremely successful, although the small size of the vocabulary (49 words) is cause for reservation on this account.
For these reasons, the system of Bowden et al. was chosen as the most promising method. In the next chapter, I will describe how this method was applied to Dutch sign language recognition, and what the results were.

Chapter Three: Dutch Sign Language Recognition

3.1 Introduction

The purpose of this project is to investigate which is the best way to perform automatic recognition of Dutch sign language, and to implement this best method in part. In the previous chapter, various techniques of recognising sign language automatically were discussed, and the 'Linguistic Feature Vector' method of Bowden et al. (2004) was chosen as the most promising approach. In this chapter, I will describe how I used this method to recognise Dutch sign language (NGT).
As explained in the introduction (section 1.3), implementing the entire method would be impossible, since it would take more time than I have for this project. I can only implement part of the method. The part I have chosen to execute is the classification part. In the Linguistic Feature Vector method, features are first extracted from a sign movie to form the linguistic feature vectors. Then these vectors are used to classify signs. It is this classification part that I will implement. To obtain the feature vectors, I recorded an NGT movie and sent it to dr. Bowden, who was willing to co-operate by using his system to extract the features automatically. With these feature vectors, I could then perform classification myself.
In this chapter, I will describe the various steps of my project. I will begin by explaining the Linguistic Feature Vector method in some detail. Then I will describe how I recorded the Dutch sign language movie. After that, I will discuss the feature extraction performed by dr. Bowden on my data, and describe my own implementation of the classification process. I will then present the test results and discuss them. Finally, I will draw some conclusions about the implementation. General discussion and conclusions are presented in chapters 4 and 5.

3.2 The Linguistic Feature Vector method

The Linguistic Feature Vector is a method for automatic sign language recognition designed by Bowden et al. (2004). In this section, I will explain the method in some detail and describe its results.

3.2.1 Overview

Bowden et al. capture their data on digital video. They then use tracking to find the important body parts in the images. This creates a body-centred co-ordinate system with which they perform stage I classification (the data coding phase). This produces a feature vector, with which stage II classification can be undertaken (the classification phase). A block diagram of the recognition process is shown in fig. 5.

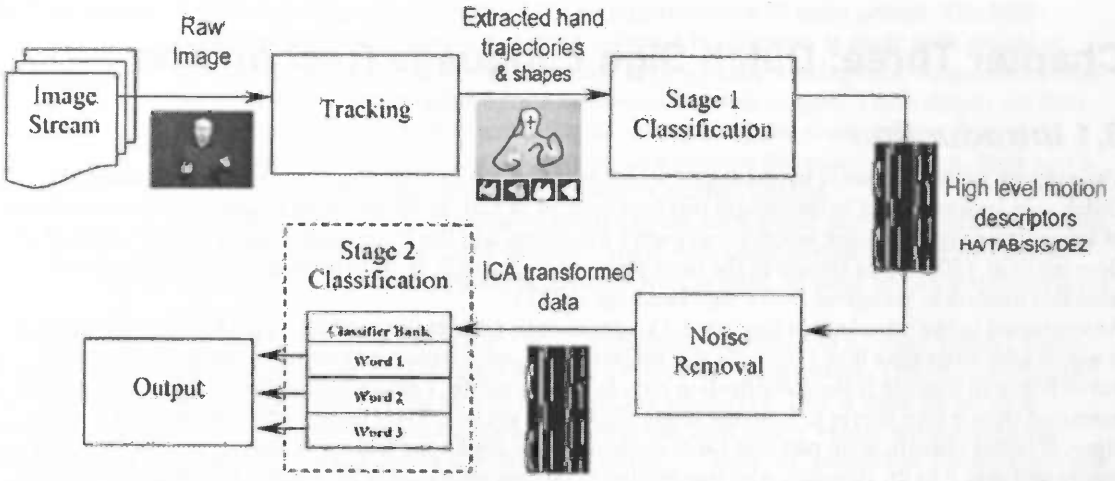


Figure 5: Overview of the Linguistic Feature Vector Method. Taken from Bowden et al. (2004)

Tracking involves segmenting the image to extract where the outline of head and shoulders is. This provides a body-centred co-ordinate system in which positions and trajectories of hands can be described.

Stage I classification converts the data (the movie frames) into feature vectors. The extracted features (also called parameters) are:

- HA: position of the hands relative to each other: 5 possible values
- TAB: position of the dominant hand relative to key body parts: 13 possible values
- SIG: Movement of the dominant hand: 10 possible values
- DEZ: Shape of the dominant hand: 6 possible values

The values of these parameters are included in Appendix I. Stage I classification results in a high-level, relatively abstract description of a sign: the frames are not encoded as 'x-position = 10, y-position = 12, x-velocity = 2.5 etc.', but as 'position = chest, movement = hand-moves-left, shape = V-hand, etc.'. Not only are these values relative to the body instead of absolute, they also generalise over small differences in position, since many different positions fall into the same category. This is a suitable way of coding signs, because small differences between position *are* not significant: it is large differences, such as differences between categories, that are significant. The features are stored in a binary feature vector (a vector of zeros and ones).

Stage II classification involves the creation of a first order Markov chain model for each sign in the vocabulary (for more information on Markov chains, see section 3.2.4). It is important to note that these chains can be fashioned from a single example, as opposed to HMMs, which require many training examples. The states of these Markov chains represent particular feature vectors, represented by symbols. If there were no inter- and intra-signer variation, then each feature vector that occurred in a sign could be used as a state in the Markov chain model of that sign. That is, if a sign consisted of the vectors (1,0,1)-(0,0,1)-(1,0,0), a Markov chain model could be made with three states, one containing (1,0,1), the second containing (0,0,1), and the third containing (1,0,0). However, this would mean that even slight variations would result in different states. Many variations, however, are the result of differences between signers, or amount of emphasis, or noise on the vector, and should be treated as one and the same vector. For example, when one signer makes a sign slightly more to the right, the TAB-parameter could be (mis)classified for a couple of frames as having value "right hip" instead of "stomach". If the actual vectors were used in the states of the model, the vectors of these aberrant frames would not match any of the states in the Markov chain for that sign. This would result in a failure to classify correctly. That is, if a second signer performed the sign mentioned above, but slightly differently, so that *his* first vector became (1,1,1), this aberrant

vector would not match the first state of the sign model, which was (1,0,1), nor any other state, so the correct model would not be found and classification would fail. So it is not advisable to use the actual feature vectors straight away as states of the sign models.
This is why the authors choose to pre-process the feature vectors before using them in the sign models. They use Independent Component Analysis (ICA) to do this. Performing ICA on a data set gives you a set of independent components and an unmixing matrix. The unmixing matrix can be used to transform the feature vectors. For the transformed vectors, 'alikeness' can be calculated, which is not so easy for the original binary vectors. ICA will be explained in more detail in section 3.2.4.
All vectors in the training set are transformed. After that, a unique symbol is appointed to each of them. the vector-symbol pairings are recorded in a lookup table. When a vector from an unknown sign needs a symbol, it is appointed the symbol of the vector in the lookup table it most closely resembles. Thanks to the ICA-transformation, this resemblance can be calculated. This use of symbols solves the problem of slight variations mentioned above: even when a vector does not exactly match the corresponding vector in the example sign, it will still resemble that one more closely than any of the others, because it is only a slight variation that makes it different. So it will receive the same symbol as that vector, and since symbols are used in the states of the model, it will now match the appropriate state, even though it was slightly different from the original vector. This way, noise due to imperfect stage one classification and variation of signs do not influence the final recognition rate as much.
Once the lookup table is created for the test set, Markov chain models can be made for each sign by simply creating a new state containing the vector's symbol for each new vector encountered in the sign. An unknown sign is recognised by extracting its features in stage I classification, transforming its vectors with the unmixing matrix and appointing symbols to the vectors by looking up the most similar vectors in the lookup table. The sign is now a series of symbols. It is classified by finding the Markov chain model that has the highest probability of producing this series of symbols. The sign represented by that model is the sign that was recognised.
The steps of the recognition process will be discussed in more detail in the following sections.

3.2.2 Tracking

Visual tracking is performed by probabilistic labelling of the skin to roughly locate the signer's face. A contour model of head and shoulders is then fitted to the signer. This outline provides a body-centred co-ordinate system. Key body locations (face, stomach, left shoulder etc.) are determined relative to this outline. The hands are found using skin labelling and their position and velocity are expressed relative to the body-centred co-ordinate system. Tracking can take place against a cluttered background and without the aid of special clothes or gloves.

3.2.3 Stage I Classification

In this stage, raw image data is transformed into a sequence of feature vectors, one vector for each frame. Linguistic evidence suggests that recognition of a sign is primarily based on the information transmitted by the dominant hand (Bowden et al., 2004). For this reason, the non-dominant hand is ignored for now and only information from the dominant hand is extracted. It is not clear how the system knows which is the dominant hand when no gloves are worn. Hand arrangement (HA), position (TAB), movement (SIG) and handshape (DEZ) are the features that are extracted. For each parameter, there is a number of possible values: 5, 13, 10 and 6 respectively. The categories are included in Appendix I. These parameter-values are determined as follows:

HA-parameter: This value can be computed directly from the x- and y-positions of the centres of the hands and their approximate area (in pixels).

TAB-parameter: This value is determined according to the proximity of the hands to the key body parts, estimated in the tracking-stage. As a distance metric, Malhalanobis distance is used.

SIG-parameter: How this parameter is determined is not described in Bowden et al. (2004). But to keep noisy motions from interfering with the detection of real movement, the size of the hand is used as a threshold for movement. This entails that small movements such as finger-wiggling and rotations of the wrist cannot be detected. This is probably the reason that all possible values for the SIG-parameter are large movements.

DEZ-parameter: British sign language handshapes can be organized into 22 main groups. The DEZ-parameter values represent six of these groups. Classification is made with the aid of exemplars. These exemplars are extracted from separate training data. Silhouettes of the six handshapes are extracted from a number of example images. These shapes are then normalized (for relative size and different orientations) and clustered into groups. Normalized correlation is used as a metric to determine distance between a shape and a cluster. A threshold determines the number of clusters formed.
Classification is now performed by extracting the normalized handshape silhouette from the image and checking which cluster it would join, using normalised correlation as a metric. For the DEZ-parameter, six clusters were created, so there are six possible values.

For each frame, these four features are extracted. The information is expressed as a binary vector. This is done by reserving one bit (a zero or one) for each possible value of every parameter. The bits representing the values that were actually found, are set to one, the others are set to zero. This is done for all four parameters. Then all four of them are concatenated to produce one final binary vector representing the information in one frame. The result is a 34-dimensional vector (5+13+10+6), containing mostly zeros and a few ones. In this manner, stage I classification translates a movie (a series of frames) into a series of 34D binary vectors. Fig. 6 gives an example of such a series of binary vectors.

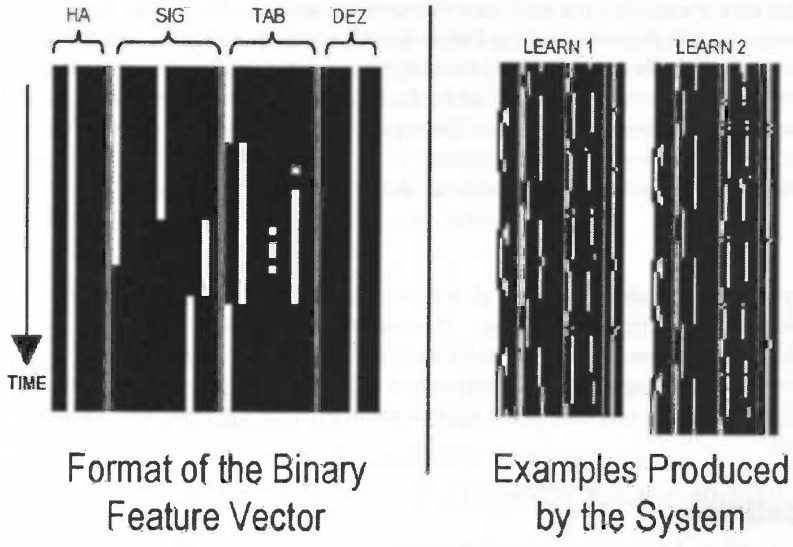


Figure 6: Example of signs represented with binary feature vectors. The vectors are depicted horizontally. Black is a zero, white is a one. The gray lines mark the boundaries between parameters. Taken from Bowden et al. (2004)

Note: within a parameter, more than one value can be active for the same frame. Whether this is a way of coding information (for instance representing diagonal movement by making both 'forward movement' and 'sideways movement' active), or whether it is a matter of values competing and both being active enough to come above a threshold (for instance when position is on the edge of 'stomach' and 'right hip' and both these values are therefore considered active), is not clear from Bowden et al. (2004).

3.2.4 Stage II Classification

After the important information has been extracted from the image data in stage I classification, stage II classification can begin. This involves building a model for each sign in the vocabulary. After this training

phase, an unknown sign can be classified by finding the model that represents it best. Before the models can be built, though, independent component analysis has to be performed on the training set, so that Euclidian measures can be used to determine how much vectors resemble each other (see section 3.2.1).

Independent component analysis

Independent component analysis (ICA) assumes that a data set is a mixed signal resulting from a number of independent sources that are mixed by an unknown mixing matrix. ICA is a statistical process that finds (estimates) the signals from these sources (the independent components), the mixing matrix and the unmixing matrix. After ICA is performed on the training set, the unmixing matrix can be used to express an original (in our case binary, but this is not necessary) vector in terms of the independent processes that produced it. In the case of the method discussed here, this means that a binary vector can be transformed into an ordinary vector, usually of lower dimensionality, by multiplying it with the unmixing matrix. Binary vectors that were the result of the same independent processes will be similar after transformation, that is, lie close together in Euclidian terms. And because noise is a process independent of information, noisy vectors will be the result of different independent components than informative vectors. This means that after transformation, noisy vectors will be far away from informative vectors in Euclidian terms. So ICA makes it possible to separate noise vectors from informative ones, and brings vectors that are alike closer together.

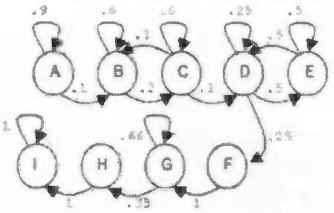
After performing ICA and finding the unmixing matrix, the binary vectors of the training set are transformed. Then a lookup table is made in which a different symbol is associated with each vector. Markov chain models can then be made using these symbols as states.

Markov chains

A Markov chain consists of a number of states with one possible observation in each state, and transitions between these states. Each transition has a certain probability. A Markov chain can generate outputs with a certain probability. It has a certain probability of starting in a certain state, and from each state, a certain probability of moving to another state (or staying in the same state), depending on which transitions there are in the current state and what their probabilities are. At each time-step, a transition is made and a new (or the same) state is reached. When a state is visited, that state's output is produced. So the probability of a Markov chain producing a certain output is the probability of it starting in the right state multiplied by the probabilities of making the transitions necessary to visit the states containing the desired output. An example of a Markov chain is shown in fig. 4.

Creating and testing sign models

When creating sign models, Bowden et al. use the symbols representing the vectors as states. Which transitions are created and how the transition probabilities are calculated is not described in Bowden et al. (2004). Judging from an example model shown in their paper it would appear that skipping states and looping between states is allowed. The example is given in fig. 7. It shows a Markov chain sign model in graphical form and in table form.



	A	B	C	D	E	F	G	H	I	J
A	.9	.1	0	0	0	0	0	0	0	...
B	0	.8	.2	0	0	0	0	0	0	...
C	0	.1	.8	.1	0	0	0	0	0	...
D	0	0	0	.25	.5	.25	0	0	0	...
E	0	0	0	.5	.5	0	0	0	0	...
F	0	0	0	0	0	1	0	0	0	...
G	0	0	0	0	0	0	.66	.33	0	...
H	0	0	0	0	0	0	0	0	1	...
I	0	0	0	0	0	0	0	0	1	...
J	.	.	.	.	.	.	.	.	.	.

Figure 7: Example of a Markov chain sign model. On the left, it is shown in graphical form: the circles are the states, and the arrows the transitions. On the right, the same model in table form. Here, letters represent the states, and the transition from A to B is shown in row 'A', column 'B'. Taken from Bowden et al. (2004)

Models are built for every sign in the vocabulary using only a single training example per sign. Then, unknown signs can be classified. This is done by first extracting the feature vectors from the sign movie containing the unknown sign, and then transforming the resulting binary vectors with the unmixing matrix. After that, the vectors are replaced by symbols. A symbol for a new vector is picked by finding the vector in the lookup table resembles the new vector most, and using the symbol associated with that symbol. Simple Euclidian distance is used as an distance metric between vectors. After the new sign is thus represented as a series of symbols, it can be calculated which model *w* has the greatest probability of producing this symbol series *s*. This is calculated with:

$$P(w \mid s) = \pi \prod_{t=1}^{m} P_w(s_t \mid s_{t-1})$$

Where

- s = symbol series
- m = length of s
- π = prior probability of the model (the sign)
- $P_w(s_t \mid s_{t-1})$ = transition probability from the current state to the next state needed

The Viterbi algorithm is used to maximise the overall probability, taking into account both the model match probability P(w|s) and the number of signs *m* that are explained, to prevent biasing towards shorter signs. The sign sequence represented by the model sequence that has the highest probability of producing the observation sequence is the sign sequence recognised.

Noise removal

After this setup was created, the authors experimented with removing noise from their data by removing certain independent components found with ICA before transforming the feature vectors. This results in a different transformation matrix, and therefore in different transformed vectors. The idea is, that noisy processes are independent from information processes. So some of the independent components must represent noise. Removing them should improve recognition.
The noisy independent components were found by trial and error. Components were removed one at a time. With each removal, it was checked whether recognition improved. The component whose removal caused the greatest improvement was discarded. Then the process started all over again, once more removing one component at a time, to find the next most noisy component and discard that one, too. It was found that the best recognition was achieved after discarding six components. Removing more components caused a drop in recognition rate, suggesting that the six discarded components represented noise processes and the rest represented information processes.

3.2.5 Results

The method was tested using a 49-sign vocabulary of randomly selected words. The dataset consisted of 249 signs, an average of five repetitions per sign. One instance per sign was used for training, the rest for testing. The recognition rate was 84%. This rose to 98% for a 43-sign vocabulary. The six signs removed were signs for which facial expression or contextual grammar were an important clue. Without removal of the noisy components (see previous section), the recognition rate was 73% for the 49-word vocabulary.

3.3 Data collection

3.3.1 Setup

I have made a movie containing NGT according to the specifications of dr. Bowden. These included using a dark background, dark clothing for the signer and differently coloured gloves on the hands. The final setup for the movie thus became as follows: the movie was recorded against a matt black background. The signer wore a brightly coloured reddish-pink glove on his dominant (right) hand, and a white one on his other hand. He wore dark clothing and stood straight in front of the camera (see fig. 8).
The person signing was a native male NGT signer (born deaf with deaf parents) aged 25. The movie was recorded with a Panasonic NV DX100 digital video camera at 25 fps.

3.3.2 Contents

The movie consisted of two parts: single words, and natural speech. In the first part, each sign in the vocabulary was made slowly ten times. Between each sign, the hands returned to the neutral position: hands clasped in front of the body. There were 31 signs in the vocabulary, so a total of 310 signs was recorded in the first part of the movie.
The second part contained a piece of natural speech: a short story of about 3 minutes. The story almost exclusively used words from the vocabulary. As it turned out, I only implemented classification of single signs, so the second part of the movie was not used in experiments.
The vocabulary words are included in Appendix II. They were chosen randomly: words with which a short story can easily be made.
Note: the word 'he' appears in the vocabulary. This is actually not a fixed sign, but one that differs with context, as is the case for all third-person pronouns in NGT (Schermer, 1991), and probably in all sign languages (Zeshan, 2004b). The word was included in one possible form (pointing to the left) as if it were a fixed sign, so that it could be used in the story in part two of the movie. This was necessary because the method implemented here cannot handle signs that differ with context, and it would have been difficult to write a story without the use of a third person pronoun. (See section 4.4.1 for more information on sign references)

3.4 Feature extraction

After the movie was recorded, it was sent to dr. Bowden at the University of Surrey. He performed automatic feature extraction (the 'data coding' step of the automatic sign recognition process) and sent the resulting binary feature vectors back to me. He reported some minor problems with the tape, which I will describe below. Furthermore, on receiving the results, I noticed that only three features had been extracted from the data, not four, as described in Bowden et al. (2004). This I discuss in the second subsection.

3.4.1 Segmentation problems

The signer's clothing was dark, but it did have some white in it, and white was also the colour of the glove on the non-dominant hand. This was a problem, because the hand tracker considers the largest area of the correct colour to be the hand. When the subject was signing, the white area of the non-dominant hand was larger than the shirt's white, so there was no problem. But in the neutral pose (hands clasped), the white hand area became so small that the white area on the shirt was larger. At those moments, the hand tracker jumped to that place. This caused bad tracking. But when the signer performed an actual sign, the tracking went fine, and since only the actual signs are used in classification, not the 'neutral' pieces in between, this tracking difficulty did not cause problems. Fig. 8 shows the signer. The same difficulty also prevented

correct tracking for parts of the second part of the movie, the 'natural speech' part. But because this part was not used in the classification process, it was again not a cause for problems.

Figure 8: Image from the sign movie. The white in the shirt interfered with finding the left hand when the hands were clasped, as they are here. During signing, however, finding the hand caused no problems.

3.4.2 No DEZ feature extracted

The feature extraction results I received differed from the theory described in Bowden et al. (2004). Only the first three features, that is hand arrangement, position and movement, were extracted, so I received no handshape information. Why this feature was not extracted, I do not know. Perhaps the images of the hand were not large (and therefore detailed) enough to allow handshape classification, which could have been solved by zooming in more.
Furthermore, the number of categories for each of these features differed from the number described in the article: for hand arrangement, there were 7 possible values instead of 5, for position 14 instead of 13, and for movement 12 instead of 10 (and of those 10, numbers 4 and 5 were skipped in Bowden et al., 2004, so that I can only provide 8 values in Appendix I). This results in a 33 dimensional feature vector instead of the 28 dimensional one described in Bowden et al. (2004). It is not possible that the missing handshape information is responsible for this extra dimensionality: firstly, the vector would then have to be 34D, since there are 6 possible handshape values, and secondly, the extraction results were given in a format that separates the three parts of the vector:

```
1       HYPER_VEC_DIM= 7 | 0 0 1 1 0 0 0
        HYPER_VEC_DIM= 12 | 1 0 0 0 0 0 0 0 0 0 0 0
        HYPER_VEC_DIM= 14 | 0 0 0 0 0 0 0 0 0 0 1 0 0 0
```

I can only guess that extra categories have been added to the feature extractor since the publication of the article. It is unclear what the values of the extra categories are. But since this information does not have to be explicit for the classification process to work, I could go ahead with implementing classification nevertheless.

3.5 Implementing classification

I will now describe how I performed automatic classification on the binary feature vectors. I will first describe the setup for my implementation and the areas in which my setup deviates from that of Bowden et

al. (2004). I will then describe the programming environment in which I worked and the actual programs. Most programming code is included in full in Appendix III.

3.5.1 Setup

The setup for the implementation is largely to follow the classification method demonstrated by Bowden et al., since it is quick, requires little in terms of training examples and produces good results. I expect this method to works as well for Dutch sign language as it does for British sign language, since the information that the feature vector contains – hand position, hand motion and hand configuration – is significant for the meaning of a Dutch sign (see section 2.1.2), just like it is for British sign language (Deuchar, 1984). So classification on the basis of this information should work just as well with NGT as it does with BSL.

The programming setup is as follows: first, the data have to be read and cleaned: text and frame number information must be discarded and the clean vectors extracted. Next, the data are divided into a test set and a training set. Then ICA is performed on the training set to find the unmixing matrix with which the binary vectors can be transformed. The training vectors are then multiplied with this unmixing matrix to produce ICA-transformed vectors. Through this transformation, vectors that are alike come to lie close together, and vectors that are different get further apart in terms of Euclidian distance (see section 3.2.4 for more information). This means that it is now possible to cluster the vectors using Euclidian distance as a metric. Which ICA algorithm Bowden et al. used is unclear. I used the FastICA program created by the ICA research group of the Helsinki University of Technology (Oja, 2003).

Next, the vectors are clustered. This is done using k-means clustering with a Euclidian distance metric. It is unclear whether Bowden et al. perform any sort of clustering. It could have been left out, but it seemed advantageous to me to perform some sort of clustering before appointing symbols to the vectors, to make sure only truly different vectors received different symbols, and vectors differing only slightly were grouped under one symbol. I used a k-means clustering algorithm created by Frank Dellaert (Dellaert, 2004). A lookup table is then created, assigning a symbol to each cluster centroid vector. Numbers are used as symbols. The lookup table is a simple table associating each centroid vector with a symbol.

Finally, Markov chain models are created from the training set using the symbols from the lookup table as states. A strict model creation was chosen: the only transitions allowed from a state were back to itself (auto-transition) or forward to the next state. Skipping states, backward transitions and loops were not allowed. The transition probabilities are determined by the number of auto-transitions in the example signs: if the example sign has three auto-transitions and then moves to the next state (e.g. a sign consisting of the symbols 22224, where three of the four transitions are auto-transitions), the probability of the auto-transition in the resulting model becomes .75, and the probability of the transition-to-the-next-state becomes .25. The probabilities in the final state of a model are determined as follows: if there are any auto-transitions in the example (as in 1244 and 12444 etc.), the probability of auto-transition in the model becomes 1, if there were no auto-transitions (as in 124), the auto-transition probability becomes 0.

When models of every sign have been created in this fashion, they can be tested. The vectors of an unknown sign are first transformed with the ICA unmixing matrix. Then for each (transformed) vector the closest vector in the lookup table (in Euclidian terms) is found and its symbol used as the symbol for that new vector. The unknown sign, first a series of vectors, thus becomes a series of symbols. Then, for every model the probability of that model producing this series of symbols is calculated. This can be done in a straightforward way, since there are only thirty-one signs and there is only one possible path through a model for a certain symbol series. The model with the highest probability is the sign recognised. Recognition rate is then calculated simply as the percentage of signs that was correctly classified.
These were the steps that were implemented. Before I discuss their implementation in detail, I will discuss some differences between my setup and that of Bowden et al. (2004).

3.5.2 Deviations from Bowden et al. (2004)

In the previous sections, I have describes the method of Bowden et al., and my programming setup to implement the classification part of this method. Before I turn to the practical part, however, there are some points that should be addressed. At certain points, I have chosen different criteria for my implementation

than Bowden et al. Also, some finer details of implementation were simply not described by them, leaving me to make a choice that might differ from the original setup. Finally, part of Bowden et al.'s method was omitted by me. I will describe these matters below.

Different choices

When building Markov chain models, one has to decide which transitions to allow, and why. Allowing loops and skips in a model can lead to unfavourable results, because a model can then give good results for state sequences it was not intended to represent. For example, by allowing a skip from the first to the last state in a model A->B->C->D, the model will be able to produce the series A-D, which is very different from the A-B-C-D series it is intended for. For instance, imagine a sign HALLO (hello) which is made by alternating three hand positions: left, middle and right (imagine a simple 'hello' wave). If the feature vector for the left position were called 'A', for the middle one 'B' and for the right one 'C', the model would become for instance the sequence A->B->C->B->A. This series represents the sign HALLO. If skipping states is allowed, though, it is possible to make a transition from the first state to the last, skipping states B, C and 2nd B. It is now possible to go from state A to state 2nd A. Both these states represent the left position. Now if someone made a different sign, say STOP (stop), which consists simply of holding the hand in the left position, this would be translated as a series A-A. But our HALLO model is perfectly capable of producing this series, by starting in the first state and moving to the last state via the skipping transition. Does this mean that the sign made was HALLO? No, it misses the necessary vectors in between. But by allowing state-skipping, these necessary vectors are not necessary any more. This is why skipping and looping can lead to unfavourable results: models can start explaining observation sequences they were never intended to explain.
In small ways, though, skips and loops can be desirable. Some signs are cyclic, that is, they consist of a certain series of vectors which may be repeated as many times as the signer wants, without it changing meaning. Imagine the sign HALLO (hello) again: waving from left to right. It does not matter how often you go from left to right and back, it is still the same sign. But for a model, this is difficult. You would need many versions of a sign model: one with one repetition: ABCBA, one with two, ABCBABCBA, etc. It would be easier to allow a *loop*, that is, to allow the model to go from the last state either to end-of-sign or back to the first state. That way, the model A->B->C->B->A could produce any number of repetitions. And all these variants *are* the sign HALLO, so it is correct that this model can produce them all.
How to decide which transitions to allow, and how to create automatically only the appropriate ones, is not a simple matter though. More will be said on this subject in the general discussion (chapter 4).

Bowden et al. do not explain which choices they make for their models, but from an example it would seem that they do allow skips and loops (see fig. 7). But to what extent this is allowed is unclear.
I have chosen to implement more strict models, demanding that every state in a model is used in the correct order. That means that skips, loops and backward transitions are not allowed. This makes a model less flexible, but it guarantees that a model only reacts to state sequences it was intended for, and not to others. This requirement was taken from automatic speech recognition theory. It is the question, of course, how well suited this requirement is for sign language recognition. This matter will be addressed in the general discussion (chapter 4).

A second difference is that I chose to build two models per sign, where Bowden builds only one. This was done mostly to reduce the chance of recognition failure due to a noisy model. A model is built from one sign instance. If the training example would happen to be a very noise one (one where the signer moved sloppily or where there was noise on the videotape), it would result in a model none of the other, 'normal' instances would match. Building two models per sign reduces the possibility of such a coincidence. Because I had less time to remove noise from my data and perfect my method, I felt that this compensating measure was justifiable. Another reason for using two instances per sign for training was to enlarge the training set, so that the results of the (statistical) ICA-process would become more reliable (with two instances per sign, my training set now consisted of 62 signs). It makes no difference for model building, though: the model is still built on the basis of a single example, only I build two models per sign instead of one.

Unclear issues

There are two steps in the recognition method used by Bowden et al. that are unclear to me. The first is which independent component analysis algorithm is used by Bowden et al. There are different ways of executing this statistical process, and which way is used could influence results. I have chosen to use the FastICA fixed-point algorithm, developed by the Helsinki University of Technology (Oja, 2003). This algorithm is quick, stable, and it was available for Matlab, the environment in which I developed my classifier.
The second unclear issue is whether Bowden et al. use any form of clustering before they build a lookup table from training vectors. Clustering is mentioned in the article, but not actually described, so it remains unclear whether it is actually performed or not. It made sense to me, though, to cluster the vectors before creating the lookup table, to avoid many different symbols for minutely different vectors. So I opted to use k-means clustering on the ICA-transformed vectors. I chose k-means, because this is a simple and reliable way of clustering vectors, and there was no reason to use a different algorithm, such as k-median.

Omissions

Due to the limited time I had for this project, I was unable to de-noise my data the way Bowden et al. did. After performing ICA and finding the independent components, Bowden et al. use a trial-and-error method to isolate those independent components that constitute noise (see section 3.2.4). Their method is effective, but very time consuming, so I did not have time to use it. I therefore omitted this part of the process.

3.5.3 Data format and Programming environment

Implementation took place in Matlab 6.0. Only the first step, reading the data cleanly, was programmed in C++, because I had more experience with text file manipulation in this language. Matlab has the possibility to interface with C++ programs, so there is no problem with using a C++ function in a fully automatic system.

Data

I received the data from dr. Bowden in the following format:

```
1     HYPER_VEC_DIM= 7 | 0 0 1 1 0 0 0
      HYPER_VEC_DIM= 12 | 1 0 0 0 0 0 0 0 0 0 0 0
      HYPER_VEC_DIM= 14 | 0 0 0 0 0 0 0 0 0 0 1 0 0 0

2     HYPER_VEC_DIM= 7 | 0 0 1 1 0 0 0
      HYPER_VEC_DIM= 12 | 0 1 0 0 0 0 1 1 0 1 0 0
      HYPER_VEC_DIM= 14 | 0 0 0 0 0 0 0 0 0 0 1 0 0 0

3     HYPER_VEC_DIM= 7 | 1 0 0 0 0 0 0
      HYPER_VEC_DIM= 12 | 0 1 0 0 0 0 1 1 0 1 0 0
      HYPER_VEC_DIM= 14 | 0 0 0 0 0 0 0 0 0 0 1 0 0 0
```

The leftmost number is the frame number. "Hyper_vec_dim" indicates the dimensionality of the vector part that follows. The three rows of zeros and ones are respectively the HA, TAB and SIG parts of the feature vector of that frame. For further processing, everything but the binary data is removed and the three vector parts are concatenated, so that the result is a matrix of zeros and ones. If the above three frames were processed that way, the result would be:

```
0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 1 1 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
```

In this manner, every sign (a series of frames) becomes a binary matrix containing the feature vectors as row vectors. These matrices are then used in the classification process. Matlab is especially suited for handling matrices, so it was the obvious choice for developing the classification functions.

There were 10 instances of each sign in the original movie. But because the process of transcribing the start- and end frames of the signs was very time-consuming (they had to be found by hand, then entered into the computer), only the first six instances of each sign were used. The first two were used for training, the remaining four for testing.

The Matlab environment

Since almost all programming was done in Matlab, it is useful to explain a few characteristics of this environment. Matlab is a mathematical environment in which matrices can be manipulated very easily. There are several data types in Matlab, such as integers, floats, characters and strings. Ordinary matrices are square and can only contain one kind of data. But there is a special data type called 'cell', which allows the user to build a matrix with different data types in it. Since cell matrices are used quite often in my implementation, it is important to understand what they are.
'Cell' is a Matlab data type for 'anything': in a cell, data of any type can be stored, and every cell in an cell matrix can hold a different data type if necessary. So when you create a cell array, you can store a number in the first cell, a string in the second, a matrix of numbers in the third, a cell array of cell arrays in the fourth, etc. Cell matrices can have any dimensionality: 1 x 1 (just one cell), 1 x N (a cell array of N cells), M x N, M x N x O, etc. Cells allow you to put different kinds of data into the same matrix. In some of my programs, cell matrices are used as tables, for example by storing model names (strings) and their probabilities (numbers) in two columns of a cell matrix.

3.5.4 Algorithms

In this section, the programs that were built to implement the setup will be discussed. Programming code of the most important functions is included in full in Appendix III. With the exception of the read function, all programs were written in the Matlab programming language. The read function was written in C++.

Read function

The C++ read function transforms the data into a suitable format. As described in the section on data (3.5.3), the feature vector data originally have the form of a text file, in which text surrounds the actual data. The read function removes this garbage, so that only the pure data is retained. The read function is given in Appendix III, section 1.

Independent Component Analysis

Independent component analysis (ICA) is used to transform the binary vectors into regular vectors in such a way that similar vector come to lie close together, and different vectors get further apart (see section 3.2.4 for more information). ICA is performed on the clean data vectors created by the read function. The mixed independent processes that resulted in the data set are estimated. This results in an unmixing matrix. With the unmixing matrix, the data vectors are transformed. After this is done, it is possible to cluster vectors, and training and testing can begin.
Independent component analysis is performed using the FastICA algorithm (Oja, 2003). The algorithm iterates until it can estimate an independent component, then moves on to find the next component. Sometimes, independent components do not converge, however, - that is, even after 1000 iterations, no estimation of a next component could be made. In that case, the algorithm fails. Such components can be converged forcibly, though, by using the 'stabilise' option of the FastICA program.

k-means clustering

After the data vectors are transformed with the ICA unmixing matrix, they can be clustered. I used k-means clustering to group similar vectors together (see section 3.5.1 for the advantages of clustering). I performed k-means clustering with the Matlab Clustering Package v.2 (Dellaert, 2004). The preferred number of

clusters can be passed as an argument to this function, though sometimes, some clusters may become empty. The program initialises the cluster centroids with a random selection from the vectors in the training set. It then assigns each vector to the cluster of which the centroid is closest to it in Euclidian terms, and then recalculates the centroid as the average of all vectors currently in the cluster. It does this for the whole data set, again and again, until no new assignments of vectors to clusters are made (that is, every vector stays in the cluster it was in already). At that point, convergence is reached and the clusters are outputted. Some algorithms execute this entire process multiple times, each time with a different initialisation of cluster centroids. The algorithm I used did not have that option, though. To avoid a bad clustering due to an unfortunate initialisation of clusters, I performed a number of runs by hand and chose a clustering in which a reasonable number of clusters converged. To completely automatise the process of sign recognition, though, an algorithm that can do this automatically must be found or created.

Building the lookup table

Next, vector quantisation is performed. This is done by building a lookup table (lut): a table associating vectors (cluster centroids) with symbols. The lookup table is implemented as a two-column cell matrix (for an explanation on cells, see section 3.5.3). The vectors are stored in the first column, and their associated symbols in the second. Anything could be used as a symbol, but for easy automatic generation I used simple numbers. An example of a lookup table is given in table 2. A lookup table like this is created from the results of clustering, and is used by many other functions, such as model-building ones, to find the appropriate symbols for vectors.
The `lut` function code is given in Appendix III, section 2.

Table 2: Example of a lookup table. Each vector is associated with a unique symbol. Vectors are not binary any more, because the lookup table is built for vectors that have been transformed by the ICA unmixing matrix, thereby becoming regular vectors.

Vector	Symbol
(-0.72, -1.88, 0.88, -0,50...)	1
(-0.72, -1.88, 0.79, -0.49...)	2
(0.91, -1.86, -1.92, -0.30...)	3
(-0.56, -1.78, 0.70, -0.50...)	4
etc.	etc.

Generating symbol lists

To facilitate model building, signs which are series of vectors have to be translated into series of symbols. The lookup table (lut) described above is built for this purpose. The `symbol` function performs the vector-to-symbol translation with the aid of the lut.
First, the function `findsymbs` is used to look up the appropriate symbol for each vector in a vector series. It does this by checking a new vector against all vectors in the lut, and picking the symbol associated with the *lut* vector that lies closest to the new vector in Euclidian terms. A sign is thus translated into a series of symbols such as (2,2,2,5,5,6,2,2,5).
`Symbol` then transforms this series into a handier format: it stores a symbol series in a two-column cell matrix. In the first column, a symbol is stored, and in the second, the number of times this symbol is repeated *in a row* is noted. So the above symbol series would be stored as: (2: 3 repetitions; 5: 2 repetitions; 6: one rep.; 2: 2 reps.; 5: one rep.), or, in everyday language: three 2's, two 5's, one 6, two 2's, one 5. Note that no information on symbol sequence is lost. `Symbol` merely stores the number of repetitions in a row as a single figure, instead of as a repetition of symbols. This makes further processing of the symbol series easier. An example of a symbol table is shown in table 3.
The function code of the symbol functions is given in Appendix III, section 3.

Symbol	Number of occurrences in a row
7	3
2	1
7	1
5	2
1	2
8	1

Table 3: Example of a symbol table. This is the symbol table for the series (7,7,7,2,7,5,5,1,1,8). For each symbol, the number of times it occurs in a row in the original series is recorded.

Creating models

In the training phase, a Markov chain model is created for each sign in the training set. A model is created from a single example. Models can be created easily from the symbol tables described above. The symbols in the first column become the states of the model. The transition probabilities of the model reflect the transition frequencies in the example sign. For instance, if a sign consists of the symbols 1224, then the transition probability from 1 to 1 (auto-transition) will be 0.0, from 1 to 2, 1.0, from 2 to 2, 0.5, from 2 to 4, 0.5 and from 4 to 4, 0.0. The Markov chain models are stored in table form (see section 3.2.4).

An example of some models is shown in fig. 9. In the first row and column of a model matrix, the states are shown. First row and first column are therefore always identical in a model. The states in the first column represent the start of a transition, those in the first row, the end. The probability of moving from state A to state B can thus be found in the cell whose row contains A and whose column contains B. So in the second model of fig. 9, the probability of going from state '5' to '8' is 0.5, of going from that '8' to '7' 0.1, etcetera.

After the probabilities are calculated from the symbol table, a Markov chain model is built by creating a matrix, putting the symbols from the symbol table in the first row and column, and putting the appropriate transition probabilities in the correct cells of the matrix. A whole series of models is created in this manner and stored in one great cell array.

An example: if a sign consists of the symbols (1,1,1,9,9,3,3,9,9), then the symbol table returned will become (1:3 ; 9:2 ; 3:2 ; 9:2). The transition probabilities become:

transition:	probability:
1-1	0.67
1-9	0.33
9-9	0.5
9-3	0.5
3-3	0.5
3-9	0.5
9-9	1.0

Making a model is now simply a matter of creating a matrix, putting the sequence (1,9,3,9) in the first row and column, and putting 0.67 in the cell with the co-ordinates (1,1), 0.33 in cell (1,2), 0.5 in cell (2,2), 0.5 in cell (2,3) etc.
Note that though there are two different '9-9' transitions, there can be no confusion between them. Although both the 2nd and the 4th state contain the symbol '9', they are nevertheless different states because their context is different: one can only follow a '1', the other can only follow a '3'. Because skips and backward transitions are not allowed in the model, there is never a chance of both '9-9' transitions being possible at the same time. This matter will be discussed in more detail at the end of the next sub-section. The function code of the modelling functions is included in Appendix III, section 4.

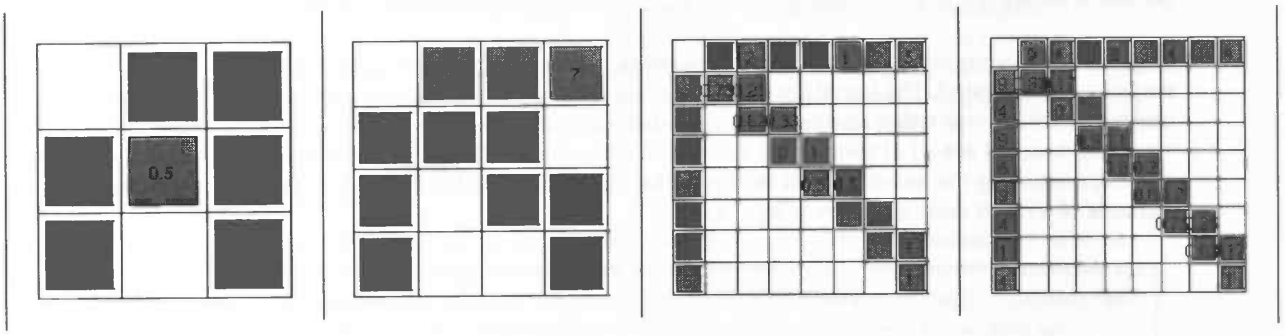


Figure 9: Four examples of table-form sign models: NIET (not), KANTOOR (office), ZIEK (ill), en PRATEN (to talk). The row and column indices are the symbols of the states of the Markov chain. The transition probability from the first to the second state can be found in cell (1,2) of the table. So in the first model, the probability of going from the first state (with symbol '2') to the second (with symbol '4') is found in cell (1,2): 0.5.

Testing models

Creating models is the training phase of the classification process. Once the models are made, they can be tested. If a new, unknown sign must be tested, it must first be transformed from a binary to an ordinary format by transforming it with the ICA unmixing matrix. Then, the transformed vectors can be translated into symbols. Now the unknown sign is a series of symbols. The probability of each model of producing this symbol series can then be calculated. If the model with the highest probability is the correct sign, the new sign has been classified successfully.
ICA transformation is performed with the unmixing matrix that was the result of performing ICA on the training set. No new ICA is done. The same goes for finding the symbol lists: the `symbol` function simply finds the closest vector in the lookup table (*lut*) for each new vector and gives it the associated symbol, no new k-means clustering is performed.
In this manner, a new sign is transformed into a series of symbols. Then, each model must be checked and its probability of producing that symbol series calculated. This is done by linearly checking each model and sorting the results according to probability.
The ICA-transformation of the vectors of unknown signs was done separately, for reasons of convenience. However, this step could have been included in the test function by adding the ICA unmixing matrix as an extra input argument and multiplying all new-sign-matrices with this unmixing matrix at the start. As it is now, the test function receives new-sign vectors that are already transformed.

Probabilities are calculated by checking the symbol series of the new sign against each model in turn. This is feasible, since there are only 62 models (2 per sign). The probability of a model producing a sign is calculated by multiplying the transition probabilities of the transitions necessary to visit the states with the correct symbols.
Models can only be traversed from begin state to the next state, and the next state, etc., to the end, because skipping states or going back is not allowed. So for a model to be able to produce a certain symbol series, it must have the exact same symbol sequence. The symbol sequence is the sequence of different symbols, regardless of how often in a row symbols occur. Thus, the symbol sequence of both 1223 and 11233333 is simply '123'. The first column of a symbol table is the symbol sequence (see table 3 for an example of a symbol table). If a model's symbol sequence does not match the sequence of a symbol series, it can never produce that series, so its probability automatically becomes zero.
Since the symbol sequences turned out to vary a lot, even symbol sequences of instances of the same sign, a matching symbol series almost always immediately meant the highest probability, since all other models would almost always have a probability of zero due to differences in the symbol sequence. For that reason,

the actual multiplication of transition probabilities was not implemented, and calculation of probability became a binary process of checking whether the symbol sequences matched or not.

The results of testing were outputted in the following manner: for each test sign, the probability of all models was calculated. The best fifteen of these were returned, in descending order of likelihood. The number of fifteen was rather randomly chosen, what mattered was that a few of the most probable models were returned, but not all of them, since this would make the results less surveyable. One big table was then created, containing the tested signs in the first column, and their probability lists in the second. A schematic example of a result table is shown in fig. 10.

sign:	probability list:	
IK (I)	'ik' (I):	0.98
	'oud' (old):	0.53
	'kletsen' (to chat):	0.50
	etc.	etc.
WERKEN (to work)	'werken' (to work):	0.88
	'kantoor' (office):	0.88
	'ook' (also):	0.76
	etc.	etc.
KANTOOR (office)	'kantoor' (office):	0.90
	'werken' (to work):	0.72
	etc.	etc.
Etcetera...	Etcetera...	

Figure 10: Schematic representation of a result table. Each sign tested has associated with it a probability list with probabilities for all models.

The success rate of the method can be calculated on the basis of a result table. This was done by simply checking for each test sign whether in its probability list, the model with the top probability was indeed the model representing the sign that was made. If so, that test sign was classified correctly. If there was a tie between different models that all had the same top score (such as for WERKEN (to work) in fig. 10), classification was counted as correct as long as one of these models was the correct one. More on this will be said in section 3.6. The total success rate was calculated as the percentage of signs classified correctly. Even with probabilities of one and zero only there were ties: some signs had models that were exactly identical, because they only differed in handshape for instance. The handshape parameter was not extracted from my data, making such signs indistinguishable to the system. However, ties occurred more frequently when another comparison criterion was used to compare a sign to the models. I will describe this alternative comparison method in the next section.

The program code for the functions described in this section is included in Appendix III, section 5.

On confusing transitions

In the case of actually calculating the probabilities of models, would confusion not occur when there are several identical transitions in a model? For example, the symbol series 121212 results in a model with three transitions 1-2 and two 2-1's. The answer is, that there can be no confusion. In calculating probability, one must start in the start state of the model (the first line of the model table). From there, the appropriate transition must be found. The only possible transitions are the auto-transition, and the transition to the next state. This holds true for all states. Since you can never skip states or go back, there is never a chance of two possible transitions at the same time. Because, although the same symbols are used more than once, the

states are still different. So in the case of the model 121212, although the first, third and fifth state all contain the symbol '1', they are still different states. There can be no confusion in calculating the probability of a symbol series, because you must start in the first state, and from there you can only stay in the first, or go to the second; from the second, you can only stay in the second or go to the third, etc. There is never the possibility of using the transition 1-2 'from the third to the fourth state', when you are actually in the first state.

Comparing symbol sets

Since instances of the same sign often had different symbol sequences, resulting in a probability of '0' for the correct model (and usually for all other models, too), I sought an alternative method of determining the model with the highest probability for an unknown sign. Though they often differed, symbol sequences of the same sign usually were *alike*, just not exactly equal. I tried to test this 'alikeness' by writing an alternative `scompare` function, which compares the similarity of symbol sequences in a very crude way.

`scompare2` performs a comparison on the basis of symbol sequences of a model and a new sign. It calculates the 'alikeness' score as follows:

$$score = \frac{N - lengthdif}{length}$$

with

- N = number of symbols in model occurring in the symbol sequence of the new sign;
- length = length of the symbol sequence of model;
- lengthdif = absolute difference between length of symbol sequences of model and new sign;

So score gets better when more of the symbols of the new sign occur in the model symbol sequence, and gets worse when there is a great difference in lengths. Notice that absolutely no demands are made as to the sequence of the symbols, this method purely compares the *sets* of symbols. This makes this comparison method so crude.
By replacing `scompare` (the binary comparison of symbol sequences) with this `scompare2` in the function `stest`, probability matrices can be generated on the basis of this criterion instead of the old one. The rest of the test method remains the same. The program code for `scompare2` is included in Appendix III, section 5.

On model names

The name of a model or a test sign must somehow be preserved; that is, after identifying which model has the highest probability of producing a certain test sign, one must be able to look up which sign that model represents. The surest way of doing this would be to store sign and model in the form of a tupel <name, matrix/model>. Because this format is cumbersome, though, I used a shortcut during my project: I always provided the signs – models and test signs – in the same order and used a simple function `names` to look up the correct name for the n^{th} sign. In the programs that have to identify models as certain signs (the test functions mostly), the number of instances per sign is often passed as an argument to facilitate this function: when the number of times each sign occurs in a row is taken into account, `names` can find the correct name whether you use one instance of each sign, or two, or four, etc., as long as the instances are in the correct order all the time. This is of course not an option for an actual system, just a shortcut I used for convenience. The `names` function code is included in Appendix III, section 6.

3.6 Results

The sign recognition experiment was performed on a data set of six instances of 31 random signs. The first two instances of each sign were used for training, the last four for testing. With the two training instances, two models were built for each sign.

First, independent component analysis was performed on the training set. It did not converge naturally, but when forced to converge, using the 'stabilize' option of the algorithm, 21 independent components were found. The original 33D binary vectors could now be transformed into 21D ordinary, non-binary vectors. Next, k-means clustering was performed. A Euclidian distance metric was used, and several iterations with different initializations were performed to find a good clustering. In the end, the data were divided into 9 clusters. Then, a lookup table was built and two models for each sign were created. The models could now be tested.

The models were first tested using the training set, to see if any information was lost during the modelling process. The expectation was, that testing on the training set would produce a near-perfect classification. This turned out to be true: both for the test of strictly equal symbol sequences and for the simple symbol comparison test, the recognition rate was 100% for the training set (62 signs).

Next, the models could be tested using the test set. The results were as follows: when the demand of strictly equal symbol sequences was upheld (i.e. when the function `scompare` was used as the comparison function), the recognition rate was 35% for a 124-sign test set. There were two models for each sign, and recognition was deemed correct when at least one of the top scoring models was the correct sign. Since there were cases in which different models (that is, models representing different signs) had the same top score, the recognition rate would have been even lower if these cases had been rejected. However, since those top scoring models were often totally identical – for instance models representing signs which only differed in handshape, a parameter which was not extracted for my data –, it can be debated whether it would be fair to the system to reject those cases.

When equal symbol sequence was not required, and a simple symbol set comparison method (the `scompare2` function) was used instead, recognition rate rose to 65%. However, once again recognition was counted as successful when at least one of the models with the highest score was the correct one. Ties occurred frequently in this setup, due to the simple nature of the comparison method, so the score would be lower if only unique identifications were counted as correct.

3.7 Discussion

The results of my classification are low compared to the results Bowden et al. obtained with the same method: 35% versus 84%. There are a number of factors that could explain this difference. Here, I will discuss the technical issues that could cause the difference. On the basis of these issues, I will draw some conclusions about the implementation in the next section. An evaluation of the entire method is presented in the General Discussion (chapter 4).
Before addressing the technical issues, a short note on the absence of handshape information in my data. Is this not a source of lower results? After all, since Bowden et al. have access to handshape information in addition to the other features, it is only natural that their classification is more successful. It would appear that this is not true, because in earlier work, Bowden et al., too, used only hand position, hand movement and hand arrangement (Bowden et al., 2003). They reported recognition rates of 100% for a 21-word vocabulary, a result comparable to the maximum 98% they achieved with all four features (Bowden et al., 2004). So maybe results similar to those described here can be obtained even without handshape information. On the other hand, it is possible that Bowden et al. did not use signs which could only be distinguished by handshape in their earlier work. Since this is not described, it remains unclear how well a random set of signs can be classified without handshape information. We will now look at some technical issues that could explain the difference.

3.7.1 No de-noising

One possible cause for my low results lies in the fact that I have not performed noise-removal on my data, as Bowden et al. did, as explained in section 3.5.2, because my limited time did not allow for this. But even when Bowden et al. did not use noise-removal, they achieved a recognition rate of 73%, which is still significantly higher than my 35%. So noise cannot be the sole reason of the difference in success.

3.7.2 Different algorithms

As I explained in section 3.5.2, there are some points on which my implementation differs from that of Bowden et al.: I am not sure which ICA algorithm they use, or which clustering algorithm, or even if they perform clustering on their data at all. If I used different algorithms for these processes, it could influence the effectiveness of the models that are built, and with it, the success rate. This can be quite influential: if for instance my ICA method is not a good one, then the ICA transformation might not lead to similar vectors coming closer together, as it should do. Then, clustering is rather pointless, and vectors that were similar originally will nevertheless receive different symbols (while vectors that were different originally may lie close together and become one cluster and receive the same symbol). Because the symbols are different, the vectors of an instance of a sign will not match with the states of the model for that sign, and classification will fail. Though I do not assume that my ICA method is so bad, I cannot be sure how different it is from the method used by Bowden et al., since I do not know theirs. The same goes for the clustering algorithm: a better clustering method can lead to better distribution of the vectors, and therefore to better models.
Because I do not know the algorithms used originally, I cannot estimate how much of the difference in success is due to these factors. However, I do not think that they are the main reason for the difference: that, I feel, lies in the fact that I build my models according to stricter rules than Bowden et al. I will explain why in the next section.

3.7.3 Skips and loops

As explained in section 3.5.2, I built my models according to the strict rule that every state must be visited, and must be visited in the right order. During training and testing, though, it became obvious that a 'wrong' vector often sneaks into an otherwise identical vector series. This could be due to variation in signing, or even to noise. If it were an option to skip such a state, the model would work. But because of the requirement of strict symbol sequence equality, skipping is not an option and the model fails. Since Bowden et al. do allow some skipping of states, this could explain why their models are more often successful than mine. An example of 'wrong' vectors messing up a model can be seen when comparing the symbol sequences of the model pairs IK (I) and WERKEN (to work) in fig. 11. In the first pair, a single '7' has strayed in amongst the '1's. The sign IK is made by pointing at the chest. One could imagine that in the second instance of the sign, the signer raised his hand a little bit higher at one moment during the pointing at the chest, and one vector consequently received the position value 'chin' instead of 'chest'. This resulted in a different vector, which received the symbol '7', whereas all the other vectors received the symbol '1'. And this causes the symbol sequences to differ. Something similar could cause the '8's that appear between the '1' and the '7' in the second instance of WERKEN. If skipping such states were allowed, the models would work more often.

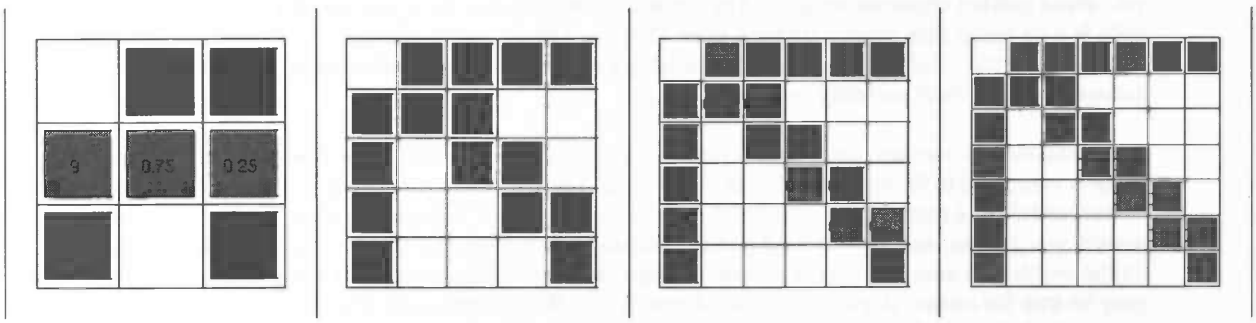


Figure 11: Two pairs of models. The first and second are instances of IK (I), the third and fourth are instances of WERKEN (to work). In each pair, an extra state causes the state sequences to differ. If it were allowed to skip such a state, models would be more successful.

The models built by Bowden et al. not only allow skips, they allow loops as well. This is another example of model flexibility which is very useful for signs. Many signs are cyclic. This means that they consist of a certain movement made a variable number of times. Since my models are linear, a model '1-2-1-2' cannot produce the series '1-2-1-2-1-2', though both these series are the same sign: a cyclic sign making first two, then three cycles. Allowing a loop helps a model to produce these different variants of a cyclic sign. For example, the models in fig. 12 clearly represent the same sign. But if it does not contain a loop, one model cannot produce different variants of the sign: the first model can *only* produce 9696969, not 9696 or 96969 or 969696 etc. But these are all variants of the same sign. Clearly it requires a model with a loop to represent all instances of such a sign.

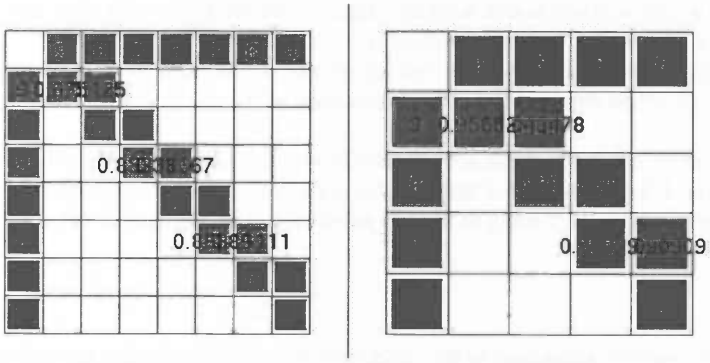


Figure 12 : Two models for VANDAAG (today). Each model can only produce the observation sequence for a sign with exactly that number of repetitions. So the second model can *only* produce 9696, not 969696 (one repetition extra). This shows why cyclic signs need models with loops.

When the requirement of strict symbol sequence equality was abandoned and a crude method of comparing the symbol sets of signs and models was used, a recognition rate of 65% was achieved without de-noising. This compares better to the results of Bowden et al., which were 73% when de-noising was not performed. This strengthens the theory that the low original success rate of 35% is mainly due to the strictness of the rule about symbol sequence equality. The extracted features seem to be suitable for classifying the signs – 65% is a lot better than chance (indeed, even 35% is). And given the crudeness of the comparison method used (see section 3.5.4) and the fact that handshape information could not be used for my data, the recognition rate could probably be higher.

In conclusion, we can say that the most probable cause of the low recognition rate is the strictness of my models compared to those of Bowden et al. The requirement that every state in the model be visited in the correct order for a model to be successful is obviously too strict. This requirement is taken from automatic speech recognition, and for this field it is a reasonable restriction. But spoken languages do not possess cyclic words. Because cyclicity is possible in sign languages, loops in models are a necessity, and the same may be true for certain skips and backward transitions. What is necessary depends on what is possible in a sign and what is not. More theoretic information on the nature and composition of signs would help to draw up better rules for building sign models.

3.8 Conclusions

In my implementation, I have followed the approach used by Bowden et al. I did not get the same results, though: my success rate of 35% was below the original rate of 84%, or even 73% if you take the recognition rate without noise-removal. The 65% I obtained with an alternative classification method

compares better to this 73%, though. Since the article describing the method of Bowden et al. is not explicit at all points, there could be technical differences between their implementation and mine which could explain part of the contrast in recognition rate that occurs when I follow Bowden et al.'s setup. However, the biggest problem is probably the strictness of my models compared to those of Bowden et al.
My model building rules are borrowed from automatic speech recognition. But certain characteristics of signs make these rules unfit for sign recognition: for instance, the fact that signs can be cyclic. With model building rules that are based on the characteristics of signs, this method could be much more successful, as Bowden et al. have demonstrated.
The difference between British sign language (BSL) and Dutch sign language (NGT) probably does not play a part in the difference in success rate. In BSL as well as in NGT, the parts of a sign that determine its meaning are: handshape, hand position, hand orientation, hand movement and facial expression (Schermer, 1991, pp. 63-64; Deuchar, 1984). Extracting information on hand position, hand movement and arrangement of the two hands and using this to classify a sign should therefore work no better or worse for NGT than it does for BSL. However, some reservation on this point may be wise. It will be discussed in more detail in section 4.6

Chapter Four: General Discussion

4.1 Introduction

The aim of this project was to find the best method to perform automatic recognition of Dutch sign language. This is a broad question, and the results can be discussed at various levels. I have looked at different ways researchers have tried to achieve automatic sign recognition. According to a number of criteria, the most promising method was chosen and a part of it was implemented. In this chapter, I will first discuss the results of my implementation of the chosen method. In the next section, I will discuss the difficulties that all current methods share and possible ways in which these could be overcome. Then, I will address some fundamental issues of sign recognition, with which most researchers have not concerned themselves yet. I will argue that it is vital not to ignore these issues, but to deal with them as early as possible in the process of constructing an automatic sign recognition system. After that, I will briefly discuss the applicability of techniques from automatic speech recognition to sign recognition. This will not be exhaustive: I will limit myself to discussing model construction, with which I have experience from this project. Finally, I will make some remarks on the possible differences between British sign language (the language the chosen method was developed for) and Dutch sign language (on which I tested this method). The final conclusions are presented in chapter 5.

4.2 Research question and results

The question I set out to answer in this project was: "What is the best way to recognise Dutch sign language automatically?". I tried to answer this question by evaluating various sign recognition methods and finding the best one. The criteria for evaluation were in part general, in part specific to the purpose of this research project, which is the eventual creation of an automatic sign language translation system. The best method was chosen according to these criteria, and it was implemented and tested for Dutch sign language.

4.2.1 The method of Bowden et al.

No current method satisfied all of the criteria specified. The linguistic feature vector of Bowden et al. (2004) was chosen as the best method because it satisfies a fair number of demands, and seems capable of expansions that would enable it to satisfy others in the future. Part of this method, the classification part, was implemented by me for Dutch sign language. Feature extraction was performed for me by the feature extractor of Bowden et al. This should not cause any problems: the feature extractor does not make assumptions about language in its processing of images. The question is whether the features extracted are as useful for Dutch sign language recognition as they are for recognition of British signs. As argued, I expected this to be so.

I did not achieve the same good results as Bowden et al. did. But there were some technical differences and problem areas which could explain why my implementation of Bowden et al.'s classification method did not achieve comparable results (see section 3.7). I was, however, able to confirm the inherent usefulness of the way features are extracted in this method by achieving reasonable results with a very crude alternative classification method.

Interestingly, I noticed some problems with Bowden's method during my project that could not be explained by the different implementation I built. I will discuss these in the next section.

4.2.2 Problems of the method

Certain signs were not handled well by the system designed by Bowden et al., and these problems seem to be inherent to the way information is extracted from video images. I will discuss four problems: three-dimensionality, the suitability of independent component analysis for this problem area, variation in start- and end position, and cyclicity of signs.

Two-dimensional representations

From the information I have on the system built by Bowden et al., it seems that no three-dimensional information is extracted from the image data. The possible values of all parameters are 2D. For position, the values describe height and width, but not how far away from the body the sign is made; for movement, there are no values describing forward- and backward motion (as in 'toward the listener' and 'toward the signer'); and for hand arrangement, no values for 'one hand in front of the other' are given (see Appendix I for an overview of the parameter values). The only possible exception is the position value 'neutral space', which denotes the space in front of the body. This could be used to make a 3D distinction in position, when the value 'chest' would mean *on the chest, touching* and a value 'chest' plus 'neutral space' would mean *chest height, but more toward the listener* (chest height in front of the body). I find it more likely, though, that this is not the case and that 'neutral space' is simply activated by any sign that occurs in that area. The fact that the position-parameter is determined from simple (x,y) position strengthens this hypothesis.
Not having any values for 3D motion, position and hand arrangement is a major deficit: it means that the difference between a sign with no movement and one with back- and forth movement in the direction of the listener cannot be detected. This is sometimes vital information: the Dutch signs SCHIP (ship) and DOOD (dead) only differ in this department (in the first sign, the handshape is moved forward, in the second it is held in place). How much of a problem it would be for the tracker to extract 3D-information is not certain, since the paper by Bowden et al. does not supply much details on how the tracker works. Perhaps it is easily expandable to analyse 3D information. If so, the system could handle this problem. If not, other solutions would have to be found, such as using a stereo camera to have access to 3D locations and motions.

Suitability of independent component analysis

While looking for a program for performing independent component analysis (ICA) on my data, I came into contact with a member of the Laboratory of Computer and Information Science of the Helsinki University of Technology, dr. Ella Bingham, who does research on ICA (personal communication, June 2004). She told me that ICA is not really suited for handling binary data, and also that it was not very reliable to use results of an analysis on new, unseen data – two things I wanted to use ICA for (for details on ICA, see section 3.2.4). This made me wonder how good the ICA transformation of unseen feature vectors actually works, and made me wonder why Bowden et al. had chosen ICA to transform and de-noise their data. Unfortunately, their paper does not shed any light on this issue. But when more information becomes available, this problem could be solved.

Variations in start- and end position and cyclic signs

During the processing of the signs I used in this project, I started noticing types of sign variation which caused the feature extractor to produce different vectors when there was really no significant difference. These types are variation in start- and end position for motion signs, and cyclic variations. I will discuss both types here.
An example of start position variation is shown in fig. 13. These are the start frames of two examples of the sign VANDAAG (today), a sign made by pointing downward twice or more. In the first example, the hand starts at about chin height, but in the second, the starting position is above head height. Since these positions are different categories, they are analysed as different values of the location parameter: the first will most likely be interpreted as 'right shoulder', the second as 'right side of face' (it is really even higher, but temple height is the highest position value available for the location parameter).
Note: I could not confirm which categories were assigned by looking at the data. The feature vectors I received had more possible values per parameter than described in Bowden et al. (2004). Because no description of the categories was provided along with the data, and because I cannot use the information in the article because of the difference in number of categories, I cannot determine which value in a vector represents which category (see section 3.4.2 about the aberrant number of categories).
Back to the difference in start position for VANDAAG (today). For some signs, the difference between 'shoulder height' and 'temple height' is indeed a significant difference in position. However, in this sign the significance lies in the pointing downward, and the height the sign starts at does not matter (though a minimum height of chest is possibly necessary). So in this case, the difference in start height is not important.

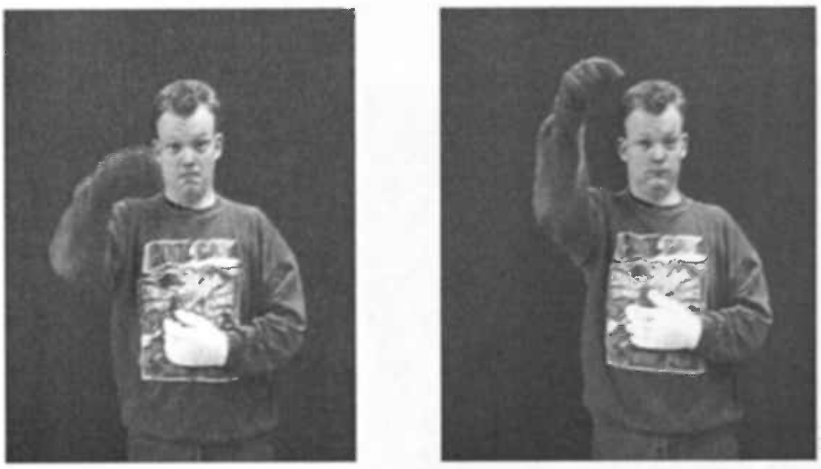


Figure 13: Two instances of VANDAAG (today). The start position of each instance is shown. As you can see, the start position is much higher in the second instance.

I observed that something similar can be the case for end positions of signs: some signs containing movement have a tendency to 'bounce' a bit at the end of the trajectory, especially when it is a sign with a motion that stops abruptly. The hand does not easily go from movement to hold, it has a tendency to sometimes 'bounce back' a little. This final movement is not significant, and also the fact that the end position can sometimes be slightly more to the left or right is not important.
So in some cases, in signs containing movement, the start- and end position are not important, and insignificant movement can occur at the end. But in other cases, and especially for all signs with no movement, (start) position does matter indeed. And of course, movement is also often highly significant. Yet Bowden's feature extractor does not make this distinction. Let us take start position as an example. The feature extractor simply extracts position from a frame in terms of a number of possible categories. So the two instances of VANDAAG shown above will be analysed as having a different start position. Probably, they will therefore result in models with a different start state. A model made from one of these instances, then, will not be able to produce the observation sequence of the other with any likelihood. So an instance of the sign VANDAAG will not be classified correctly if it has a start height different from that of the training example. But it *should* be: in this case the difference in start position is not significant. This is a problem that bears thinking about: how can you create a system that automatically detects when variations in position like this are significant, and when they are not? What exactly *are* the rules of performing signs: how much variation is allowed? And when? It seems that once a sign contains motion, the nature and direction of motion is the most important aspect, and precise position of start and end become less significant. Should we then process signs with and without motion in different ways? And even if we know all the rules of sign variation, can we build a system that can implement them? I will get back to this after discussing another example of insignificant variation: cyclicity.

A cyclic sign is a sign containing a certain circular or to-and-fro motion that may be repeated from two up to as many times as the signer wants, without it affecting the meaning of the sign. Furthermore, start- and end position are insignificant: when performing a circular motion, it does not matter where on the circle you start or end. Especially end position varies a lot, possibly under influence of the location of the next sign a signer wants to perform (if it is on the right of the location of a circling cyclic sign, stop at the right side, if it is above the current sign, stop at the top, etc.). An example is shown in fig. 14: two end positions of the sign DAAROM (that's why), which is a (from the signer's point of view) clockwise circling motion in the vertical plane. The same goes for to-and-fro signs: end position can occur anywhere on the trajectory (start position seems more fixed for these signs, though).

Variation like these could be dealt with by building a cyclic model and allowing it to start and end in any of its states, which is what Bowden et al. seem to do. However, the problem again is how to detect automatically when such a model should be built. And are start- and end position insignificant for all cyclic signs, or are there exceptions? And how can you include the requirement that a cyclic motion must be performed at least twice, if your model allows any state to be start- or end state?

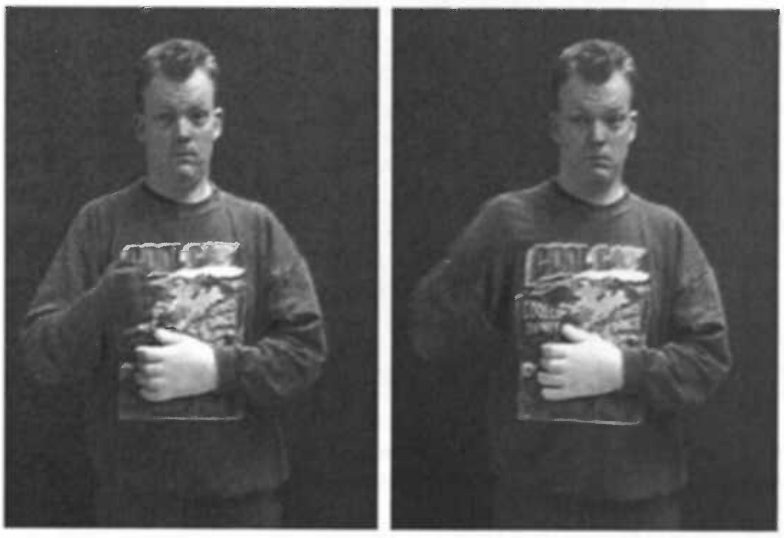


Figure 14: End positions in two instances of the cyclic sign DAAROM (that's why), a clockwise circling movement in the vertical plane. In the first instance, the hand stops on the uprise of the circle; in the second, it moves on and reaches the top of the circle before it stops.

The problem with both these types of variation is, that what is insignificant variation for one sign, is a significant quality for another (for example 'start position'). Significance of a difference of start position or end position is therefore apparently not a quality of how great the difference is, but an effect of the *kind* of sign it occurs in, and possibly even of the kind of position, movement etc. that a sign contains. To give an example: the variation in position of shoulder height or head height is in principle significant. But in some cases, such as the example of VANDAAG discussed previously, it is not. What *is* significant there is the fact that you start high, and end low. But how high exactly is not important here, though it is for other signs. So you cannot state absolutely which feature, which difference in feature value is or is not significant: it depends on the sign that is made.

Even less restricted in their features are cyclic signs: in these, it does not matter how many times you go through the cycle, or where you start or end, as long as it is somewhere on the cycle. But in an ordinary sign, even one with movement in it, (approximate) start- and end position *are* often significant. It is difficult to draft rules that say *when* variations are insignificant and when they are not, and it is equally difficult to program a system in such a way, that the correct judgement is made in all cases. Humans do it effortlessly, though. We 'see' a sign like VANDAAG as simply 'pointing downward a couple of times', so how high you start exactly is not important, as long as it is higher than where you finish. Perhaps if we coded signs in such terms, too, instead of in terms of position and movement, then these problems could be dealt with more easily.

Onno Crasborn, of the University of Nijmegen, has proposed an interesting theory in this respect: he proposes a description of sign language signs not in terms of position, handshape, etc., but in terms of which shape a signer is creating in space (Crasborn, 2003), arguing that this is a more natural way of representing signs. So when depicting a cylinder in front of the body, instead of analysing it as: 'two C-hands starting in the middle and moving apart', one could simply encode it as 'horizontal cylinder'; instead

of describing the sign DAAROM as 'position chest, movement sideways, position right hip, movement downward, position stomach, movement left, position left hip, movement upwards, etc.', one could simply encode it as 'clockwise circle in the vertical plane'. Then, it does not matter any more where you start or stop, in each case you draw a circle in the vertical plane. Descriptions like these would solve the problems of variation mentioned here, but it remains to be seen whether all signs can be uniquely described in these terms, and whether automatic tracking can be performed at such an abstract level.

4.3 General sign recognition problems

In the previous section, I described some difficulties of the method that was chosen as the best, the linguistic feature vector of Bowden et al. (2004). The survey on sign recognition research also revealed a number of problems that almost all methods, including the one that was chosen, had in common. In this section, I will evaluate these problems. I will also speculate upon the ability of the linguistic feature vector method to deal with these problems, and upon how it could do this.

4.3.1 Tracking: finding hands in a cluttered background

Many methods I evaluated used either instrumented gloves, or required that the signer wear coloured gloves on his hands. The method of Bowden et al. is ahead of the field in this respect, because the system can track the hands using skin colour detection, and can work against a cluttered background, too. However, in my own movie I was instructed to use a dark, neutral background, dark clothing and coloured gloves, suggesting that skin-colour tracking is not completely straightforward.
But even when a system is able track using only detection of skin colour, what will happen if the signer wears clothes that are nearly the same colour, like salmon or beige? What will happen if the background has an interfering colour? These are problems that all image-based methods have in common. But even in the case of using instrumented gloves, some video material will have to be used to evaluate the non-manual component of signs. So even glove-based methods will have to deal with tracking problems sooner or later. Bowden et al. provide interesting results on tracking against interfering backgrounds in Ong & Bowden (2004). They make use of the fact that an interfering background item will not fit a handshape detector better than the hand itself, which enables them to find the hands even when the background contains items of the same colour. If something similar is possible for face recognition, then the problem of tracking could be solvable.

4.3.2 Using non-manual information

Almost all methods that were evaluated by me used only hand information to recognise and classify signs. Some only used the dominant hand, some used information from both hands (for instance Vogler & Metaxas, 1997,1999). But in sign languages, non-manual information carries meaning, too. Some signs only differ in facial expression or shape of the mouth. This is true for both Dutch, and probably for all sign languages (Schermer, 1991, pp. 63-64). For example, the Dutch signs VERVELEND (annoying) and VRIJDAG (Friday) can only be distinguished by facial expression.
It is true that many methods use only a subset of the important features of a sign anyway: Bowden et al., for instance, do not detect orientation of the palms, and other methods do not use handshape (Vogler & Metaxas, 1997, Holden et al., 1999). But adding these features to a system is a smaller step than adding non-manual information: extra hand information is a matter of expanding the hand tracker (as Bowden et al. demonstrated by adding a handshape recogniser to their system between 2003 and 2004). Non-manual information requires recognition of facial expression and mouth shape, which requires a different kind of automatic image processing.
Using non-manual information is especially a problem for methods that use instrumented gloves and magnetic trackers, since these do not have access to information about the face, and usually do not include methods of processing visual information. But image processing is the only way to analyse facial expression: wearing electrodes on your face which can detect muscle contraction and infer facial expression from that would be rather cumbersome for the signer; besides, it is doubtful whether such a thing is even possible.
So glove-based methods are going to have a problem adding non-manual information to their systems, because this requires image-based processing. Methods that already base themselves on video images can add this kind of information more easily, but it would still require a different kind of tracking. The system

built by Bowden et al. does not process non-manual information, but since it is a video-based method, this information could be added.

4.3.3 Finding word boundaries and co-articulation

All methods that wish to handle continuous sign language have to deal with the problems of finding word limits, co-articulation effects and movement epenthesis. The system built by Chen et al. can recognise sentences using whole-word models, but they do not mention how this is archived (Chen et al., 2003). Other methods (Vogler & Metaxas, 1999a; Bauer & Hienz, 2001) have tried modelling phonemes instead of words, so that they can recognise each phoneme, and piece together which parts make up which words during classification. The problem of handling continuous sign language remains difficult. Bowden et al.'s system, too, has to cope with this problem. They do not appear to test on natural continuous sign language, but they do seem to test by presenting all test signs in a row, leaving it to the system to find the best word sequence to explain the entire observation. (This differs from my approach: I simply presented one sign at a time and calculated which model had the best chance of producing this sign.) To quote their paper:

> "During classification, the model bank is applied to the incoming data in a fashion similar to HMMs. A sliding temporal window T is used to check each of the classifiers in turn against the incoming feature vectors. The objective is to calculate that chain which best describes the incoming data i.e. has the highest probability that it produced the observation sequence s. (...) .. to find the sequence of words that best explain the observations over time." (Bowden et al., 2004, pp. 8-9)

So Bowden et al. leave it to the system to decide where the word boundaries are, but they do not use natural sentences as test material: they only record signers signing single words. So although they offer signs in a row, those are still separate signs, not co-articulated ones. That means that, at the moment, their system is not equipped to deal with co-articulation effects and movement epenthesis.

4.3.4 Using large vocabularies

Every method that uses whole-word models to recognise signs ends up with a vast amount of models. A sign vocabulary consists of a few thousand signs – this differs among languages (about 6,000 signs for Chinese sign language; Chen et al., 2003). This means that, to be able to deal with natural sign language, whole-word based methods will have to create and search through thousands of models. In this respect, methods that model sign components, such as phonemes, have the advantage: there is only a limited amount of phonemes, from which an infinite amount of words can be built. Modelling all those words separately causes specific problems: to search through such a huge amount of models takes time. To deal with natural language, though, a system has to be able to work in real-time. So optimisation techniques for finding the right subset of models fast will have to be employed. Chen et al. have developed such methods: they experiment with vocabularies of about 6,000 signs, and they report a significant drop (11 times as fast) in time required (Fang et al., 2002). Since Bowden et al., too, work with whole-word models, they will need to deal with this problem also, eventually. Their current system does not contain optimisation techniques – they are not necessary yet, because only small vocabularies of 20-50 signs are used.

4.3.5 Inter-signer variability

If a system is to be of practical use, it has to be able to understand different signers. Each signer performs signs slightly differently, just as all speakers sound a bit different. A system either has to be robust enough to understand all signers, or has to have to possibility to be trained by a user to the specifics of that user's signing.

Most systems I evaluated used data from one person only. One exception is Chen et al. (2003), who used six different signers in their latest demonstration. They reported a drop in accuracy compared to using data on one signer. Their accuracy dropped even more when they tested on new, unseen signers. Vamplew & Adams (1998) reported a similar drop in accuracy when testing on unseen signers.

Bowden et al., too, used only one signer to train and test with. How to enable a system to understand any signer, with or without extra training, is a problem that remains difficult, not only in sign recognition, but also in the associated field of automatic speech recognition. As an extra problem, it would appear to be more difficult to design a training program for a system using whole-word models than for one using phoneme models. One can imagine building a learning algorithm that asks the user to perform each

phoneme, so that it can adjust its models according to the signer's characteristics. But one could hardly do the same for a whole-word system and ask the user to perform every word in the vocabulary, a possible 6,000 signs! This is a serious problem that bears thinking about.

4.3.6 Expandability of the selected method

In the previous sections, I described a number of problem areas that are difficult for all automatic sign recognition methods. The method I considered to be most promising, Bowden et al.'s feature vector, is no exception: in its current form it, too, is not equipped to deal with these problems. But one of the criteria for a valuable method is that it is expandable. Let us consider which of these problems could be solved within the framework set up by Bowden et al.

Tracking against an interfering background may simply be a case of improving the tracking methods. The results reported by Ong & Bowden (2004) certainly seem promising.

Adding non-manual information should not be too difficult for this system. It is already image-based, so it has facial information available, as opposed to glove-based methods. Of course, a facial expression recogniser has to be built. But the nature of the system is such, that this extra information could easily be incorporated: it is simply a matter of designing categories for the 'face' feature and adding these to the binary feature vector (see section 3.2 for more information on the feature vector method). So the ease with which this problem could be solved really depends on the possibilities of computer vision: how well can facial expressions and mouth shapes be analysed automatically? For other information not contained in the feature vector, such as hand orientation, the same holds true: once automatic analysis of this feature can be performed, it is easy to add the results to the feature vector. The bottleneck is computer vision, not adding the features.

Dealing with continuous sign language is more of a problem for Bowden et al.'s system. When real, natural sign sentences are used, co-articulation and movement epenthesis become a problem. If a good, video-based method for segmenting sign sentences (and perhaps filtering out movement epenthesis) could be found, then sentences could be pre-processed and the results classified with the Bowden system as it is now. On the other hand, it is more probable that segmenting and recognising will have to go hand in hand, something which Bowden et al. already practise in a limited way. In that case, extra algorithms will have to be added to deal with co-articulation effects and movement epenthesis.

If Bowden et al. want to realise their objective of dealing with large vocabularies (Bowden et al., 2004, p.2), they will need a large amount of models. To search through these takes time and resources, as explained earlier. Bowden et al. do not propose any methods of dealing with this themselves, but perhaps they could borrow the techniques developed by Chen et al. (2003) to deal with this problem. They could pick up some features early in the process, such as whether the sign is one-handed and whether in contains any movement, and use these to home in on the likely subset of models. In any case, there is no reason why such techniques could not be applied to the feature vector system, so the problem of searching through large model banks should be solvable.

Variability in the way people perform signs is a problem for Bowden et al.'s system. As explained above, individual fine-tuning is probably not easy to realise for a system that uses whole-word models. Bowden et al. try to encode a sign in such an abstract way, that individual differences are not retained by the system, so that they will not feature in the creation of a model. They do this by analysing data in terms of broad categories, thereby presumably generalising over individual differences between signers. It is not clear whether this works well enough, though, because the system has not been used with more than one signer or tested on new, unseen signers.

4.4 Fundamental sign recognition issues

The problems described in the previous sections are rather practical, and they may be solvable by developing better computer vision methods, using faster computers and designing algorithms to make time-consuming processes faster. But there are other issues for which the solutions are less easily envisaged. They are the more fundamental problems of sign recognition: signs that change with context, sign constructions that convey meaning through spatial relationships, and translating sign languages into written language. In their current form, none of the sign recognition methods described earlier are capable of dealing with these problems, and most researchers ignore them 'for now'. But if the aim is to build a real-world sign language translation system, these issues cannot be ignored forever. And I do not believe that solutions to these problems can easily be added later, the way the problems described in the previous

section can often be solved with extra modules or techniques. The problems described here are much more fundamental, and will therefore need to be considered from the start of the sign recognition process. Adding solutions later is not an option for these problems. The problems fall into two categories: variable signs and sign language translation. I will discuss both below.

4.4.1 Sign language grammar and the phenomenon of non-fixed signs

As explained in the introduction (see section 1.1.4), signs come into two categories: signs that always have the same form, and signs whose form differs under the influence of morphological or syntactical processes (the fields of morphology and syntax are not clearly separable in sign languages; Schermer et al., 1991). Especially movement is often used to define syntactical relationships in a sentence: the direction of movement of a verb sign can determine the subject and object of a sign, rather like a conjugation. There are several examples of such variations in sign form. I will discuss them briefly below and indicate why the current sign systems are not equipped to deal with this kind of signs. All current systems use only signs that always have the same form, but to be able to process natural sign language, they must be capable of processing the other category of signs also. The phenomena described below are taken from Dutch sign language grammar, but most will exist in some similar form in other languages, too (Zeshan, 2004b; Emmorey & Falgier, 2004; Nyst, 2004; Liddell, 1980).

Reference through location

In Dutch sign language, and probably in all sign languages (Zeshan, 2004; Emmorey & Falgier, 2004), referring is done by pointing. A third-person referent is placed in syntactical space (which is in front and to the sides of the body and rather larger than signing space, in which ordinary signs are made). Referring to that person is then done by pointing to that location. If one referent occupies a spot, nothing else can be placed at that spot. A referent keeps its spot until a shift in conversation takes place, which happens for instance when the subject is changed, or when location or time are shifted.
To interpret sign references, the location that is pointed towards must be retained by an automatic recognition system. Then, the reference can be resolved later, just as it can be done for text documents in ordinary natural language processing. But if the spatial information is not detected, it can never be retrieved later who or what the object of the reference was: after all, the only clue is the location in space. This is why current systems would not be able to deal with references: they only detect hand position, movement etc. and classify a sign accordingly. To deal with references, a system would have to: a) be able to recognise certain signs on the basis of a only part of their features (the non-varying features: it must be able to detect that something is a reference without using the feature 'movement direction', since this varies with context), and b) retain the location information for referents and references, so that these may be solved later when a correct sentence must be created from the information detected. Current systems just do not allow for these possibilities.

Conjugation through direction of movement

Some verb signs containing movement are conjugated by changing start- and end point (sometimes only one is variable). For example, the sign GEVEN (to give) is made by holding a flat hand palm upwards and moving it *from the one giving towards the one receiving*. If a third person is subject or object, the point where that person (or object) was located is used as start- or end point.
What was said in the previous section, goes here, too: a system must be able to recognise some signs on the basis of part of their features (for GEVEN, movement, start point and end point cannot be used, since these can vary), and it must retain information about the start- and end point of such a sign, since this carries the information about who is the subject of the verb, and who the direct or indirect object.

Incorporation of qualities

There are signs that express adjectives or adverbs, and these can be used to describe qualities of a certain noun or verb. However, in sign language, describing qualities can also be done (to a certain limit) by incorporation. So describing a big book can be done by making the sign for 'book' large, describing cycling fast can be done by making the sign for 'to cycle' quickly, etcetera. This can of course not be done for

every quality, or for every sign (signs made on the body are usually less flexible), but incorporation is possible and is used a lot.
Current sign recognition systems do not allow for this, however. If anything, they often generalise over size- and speed differences to be able to generalise over different signers. This is of course useful, but it makes it difficult to see how incorporation could be detected automatically. Often, a certain facial expression and body tilt accompanies incorporation. Checking for these might help in detecting and interpreting incorporation. At the moment, though, no system uses non-manual features, so this is not an option.

Localisation: placing objects in space to express their relationships

In sign language, persons and objects can be placed into syntactical space: this is how referring is achieved. This placing in space is called localisation. Localisation can be done by making the sign at the desired place directly, or by making the sign first, then pointing at the desired location to localise it there. Localisation is used to express relationships of referents to one another. This can be a spatial relationship: the tree stood next to the house, and in front of it was a lamppost, or it can be a semantic relationship: sketching a 'family tree' in syntactical space to explain your family relationship to someone else, giving an overview of the structure of your company, etcetera.
To pick up this kind of information, a sign recogniser would have to have a representation of syntactical space and constantly update it, the way a human listener does. Both the acquisition (detecting locations very precisely) and the processing of such data (drawing conclusions from spatial relationships to understand what was said) would be difficult tasks. But this, too, is a part of natural sign language.

Classifiers: letting a handshape assume the role of an object

Finally, there are classifiers. Classifiers are handshapes that temporarily assume the role of a referent. Certain handshapes are used for referents with certain qualities (the flat 'B-handshape' for flat objects, such as paper, the one-finger '1-handshape' for persons, etcetera). A classifier can show what a referent did or what happened to it by 'acting it out': for instance if you want to describe a car that slipped, you let your hand assume the role of that car and make a skidding movement with it.
Classifiers are very difficult to interpret automatically, because interpreting them depends for a large part on experience, expectation, imagination: being able to see a handshape as a symbol for a certain object and therefore being able to interpret the movements of that handshape the correct way: when you know the hand represents a car, you will interpret forward motion as driving; when you know it is a piece of paper, you will interpret it as sliding (across a table for instance); etcetera. It is difficult to duplicate this recognition process artificially. The fact that there is a logical relationship between referent and handshape used could help, as well as the fact that certain handshapes are always used for certain objects. But a lot of the interpretation remains context-dependent and cannot be done without thorough knowledge of the real world. It would be very difficult to enable automatic sign recognisers to deal with classifiers. Needless to say that the current systems cannot handle this grammatical structure.

4.4.2 Translating sign language into written language

This issue is related to the one in the previous section in that a lot of the constructions described there are syntactical: important for putting the recognised signs into the correct relationship to each other. When simply recognising and transcribing signs word-for-word, a lot of the information described in the previous section remains unused. Then, all but the most simple of sentences soon become obscure. A Dutch sign language sentence expressing "Jan gave Marie a big book" would result in a literal transcription such as: "JAN REFERENCE. MARIE REFERENCE. BOEK REFERENCE GEVEN REFERENCE." ("Jan reference. Marie reference. Book reference to-give reference."). Even in this simple sentence, the information of who gave to whom, and the fact that the book was big, are lost. You know that Jan and Marie were placed in space: this is expressed by the first two sentences. You know that BOEK was the object and that is was given by one referent to the other. But which referent gave and which received is not clear. This information was contained in the location of the referents: the point in space the reference sign indicated. But that information is lost when the sign is simply recognised as 'a referring sign', without retaining also the information of *where this referring sign pointed to*. In a similar way, the information of the size of the book is lost because it was incorporated in the sign BOEK. If that information is not

retained, the fact that it was a big book is lost and cannot be retrieved later. Such information is essential for the correct understanding of a sentence, though. So to automatically recognise a conversation, a story, etc. in a useful way, syntactical information must be detected and interpreted. Then, the meaning of the sign language sentence can be represented in written language. This is what translating is: expressing the same information that was contained in the original sentence in the new language. In this case: a textual form of some spoken language. For example, imagine you want to create a system that recognises Dutch sign language and outputs it as written Dutch. To be able to do this, a system needs the syntactic/morphological information described in the previous section. Since this information is often expressed spatially, sign recognition systems need some way to deal with these spatial data, to process it and store it in some way that preserves the information carried by it in the original sentence, so that it can be used to construct a written sentence containing the same information. No current system is equipped for this task, and none can easily be adapted to handle it.

To solve the problems described above, systems have to be designed to deal with spatial, grammatical information from the very first step. Features such as direction of movement and location of a referent have to be detected in the tracking phase, and preserved in some suitable format until such time as they are needed to reconstruct the meaning of a sentence. This is all the more problematic because the system cannot know what is (spatial) syntactical information and what is simply a phoneme (like 'location of the hand') until the sign is recognised. And it cannot recognise the sign without knowing its phonemes. So certain spatial tracking information has to be stored in a way that leaves it open to be interpreted either as an 'ordinary' phoneme, or as a syntactical piece of information that will be needed later. In the classification phase, it can be decided which role it plays. Top down processing will also be needed here: in case both roles are possible, the signs that have been recognised already and possibly even the piece of sentence that has already been put together will have to help decide which interpretation is more likely. All in all, the processing of syntactical and spatial information will have to be incorporated into the sign recognition process, from the tracking stage all the way through to the translation phase. Only then will there be a chance that real, natural sign language can be automatically recognised successfully.

4.5 Sign recognition and automatic speech recognition

In my implementation of the recognition process, I built Markov chain models. The restrictions imposed on these models I borrowed from automatic speech recognition, in which Markov chain models are also used. Experiments showed, though, that this did not work well: the models were too strict for sign language recognition. Speech is sequential: all phonemes follow each other, so it is only natural to demand that a model for a spoken word visit all its states and not loop back. But in sign language, phonemes can occur simultaneously as well as sequentially. This means that more variation is possible in a sign than in a spoken word. And then there is cyclicity. There are no cyclic words, but there *are* cyclic signs, and to model such a sign you need to have a loop in your model. This is not allowed by speech recognition, but indispensable for sign recognition.
So theoretical reasons as well as the empirical evidence presented in this project shows that the restrictions and rules that work well for speech recognition do not always work well in the field of sign recognition. This means sign recognition has to draw up its own rules. The rules of speech recognition may be too strict, but abandoning all rules for building models is also not desirable (see section 3.5.2). But what rules are suitable for building sign models? What is permissible in a sign model? Can models be partly cyclic (that is, are there signs that contain small, inner cycles) or is it only allowed in a sign to repeat it in total (in that case, only a loop over the whole model should be allowed)? Are there sometimes parts of a sign that are optional (this would call for skipping options in a sign model)? And how much variation is allowed in the begin- and end position of signs containing movement? Are they always allowed to bounce back a bit from their end position (in that case, an optional backward transition to one or two previous states would be useful in models of signs containing movement) and vary in height of their start position (this would call for allowing any of the first few states to be the start state)?
It is clear that more theoretical information on what is allowed in a sign, which parts can be optional, which parts can vary, etcetera, is vital for drawing up reliable rules for building sign models. Simply tinkering with your model until you get good results is not an option, since you would then be optimising for a certain data set; an optimisation which might not prove so good once the models have to deal with new,

unseen data. A solid theoretical base guarantees that the models you build will be suitable for a sign *language* (or even all sign languages?), not a sign data set.

4.6 Dutch sign language and British sign language

As a final item in this discussion, I want to make some remarks about the possible differences between Dutch and British sign language (NGT and BSL). I have argued that, since Bowden et al. (2004) use parameters for classification which are significant for the meaning of Dutch signs, too, their method should in principle be applicable to Dutch sign language. However, there is of course the matter of how many, and which, possible values you create for each of the parameters. Bowden et al. created a number of possible values for hand arrangement, position and motion which seemed logical and well applicable to NGT. But how they arrived at this categorisation, and how much of it is especially suitable for BSL, is uncertain. Perhaps those categories, that reflect the distribution of feature values in BSL well, are much less suitable for describing NGT? For example because NGT might use finer distinctions in position than BSL? In that case, the nature of the parameter categories makes the method less suitable for NGT than for BSL.
This problem becomes even clearer when we look at the handshape parameter. Bowden et al. mention that there are 57 unique handshapes in BSL, which can be divided into 22 main groups. This differs from NGT: NGT has about 70 different handshapes, although it has not been investigated yet whether they are all truly significantly different. These can be divided into 8 groups, though admittedly the last group, containing 8 shapes, is called 'miscellaneous'. This is a very different distribution: in BSL, there are on average 3 shapes per group, in NGT the average is about 9. Either different grouping criteria are used, or very different handshapes are used in both languages. Both options make it uncertain to what extent the parameter values of the handshape parameter designed for BSL are suitable for NGT handshapes. Of course, the main objective is to group similar handshapes under the same parameter value, whether this is theoretically the correct group or not. But maybe the six shapes that are chosen as parameter-value exemplars by Bowden et al. (they use only 6 of the possible 22 groups as handshape categories) cover British handshapes well, but Dutch handshapes poorly.
So even if a method designed for one language uses sign features that are significant for the signs of another language as well, it still does not mean that that method is well applicable to the other language. The feature *values*, too, must be suitable for both languages. Though the most questionable feature of Bowden et al.'s method with regard to NGT, handshape, was not used in this project, it is still important to take notice of this extra difficulty of applying a method to a different language.

Chapter Five: Conclusions

5.1 Research question

The question I wanted to answer in this project, was: "What is the best way of performing automatic sign language recognition on Dutch sign language?". I have argued that there is no reason why a method would be more suitable for one language than for another, because all sign languages seem to transmit information in a similar way, using handshape, hand motion, hand location, hand orientation and non-manual features. This has not been formally investigated and proven, but evidence from many different sign languages shows that in all of them it is these features that determine the meaning of a sign (Zeshan, 2004b). So I found it justifiable to search for a method that worked well, regardless of the sign language it was tested on, because as long as it used sign features that are important in Dutch signs, there is no reason why such a method should not be applicable to Dutch sign language. The task of finding the best sign recognition method for Dutch so became a task of finding the best sign recognition method for any language, and checking whether it used features that would also be useful for the classification of Dutch signs.
To answer this research question, I have studied the field of automatic sign recognition and chosen a method which I believe to be the most promising, and which seemed applicable to Dutch sign language. I then implemented the classification part of this method for Dutch sign language, to investigate whether this was true, whether this method could indeed be used to classify Dutch signs successfully.

5.2 Experimental results

The sign method I chose as the most successful one was the "Linguistic Feature Vector" method of Bowden et al. (2004). With the aid of dr. Bowden I used the method on Dutch sign language data. I implemented the classification phase of the method, following the method of Bowden et al. However, my classification rate of 35% was below the 73% achieved by Bowden et al. (84% when de-noising was employed). Yet I achieved a recognition rate of 65% with a crude alternative classification method, which comes closer to Bowden et al.'s result. So it is not the case that the extracted features were unfit to classify Dutch sign language signs with; rather, my implementation of Bowden et al.'s classification algorithm, which yielded the 35%, differed from theirs on some points. This probably explains the difference in success rate. Yet I also encountered some difficulties inherent to the way the system of Bowden et al. extracts features and classifies. This caused me to speculate about a different representation of signs, which would eliminate those problems.

5.3 Current limitations in sign language recognition

In my investigation of the field of automatic sign recognition, I came across a number of difficulties that almost all systems, including that of Bowden et al., shared. This method could be modified to deal with many of those problems, though, by adding modules, or designing economic algorithms for certain parts of the recognition process. But coping with inter-signer variability is a real problem: it remains to be seen whether Bowden et al.'s feature vector method generalises enough during recognition to be able to correctly classify signs from different signers. This has not been tested yet.
And then there are some issues in the field of automatic sign recognition that current researchers do not occupy themselves with yet. These differ from the difficulties mentioned above in that they are not easily solved by adding an extra module to a system. These fundamental problems are: processing signs that vary in form according to syntactical demands, and detecting and retaining syntactical information, which is needed to determine the meaning of the signed utterances. The problems are related, they have to do with the way syntactical information is expressed in sign language sentences. Often, such information is transmitted by using space: conjugation and references are but two examples of this phenomenon. If certain locations, directions of movement and spatial relationships are not detected and retained by the tracker right at the start, the information is gone and can never be retrieved again later. This makes the problem so fundamental: to be able to understand the *meaning* of a sign language sentence, not just the signs, one has to detect the syntactical information in a sentence as well as which signs were made. And understanding the meaning of what was said is vital if you want to express it in another language, that is if you want to translate sign language. To reconstruct the meaning of a sentence, one needs syntactical information. Since

this is often spatial in sign language, one needs spatial information. And this information needs to be detected immediately in the tracking phase, or else it is lost. So spatial/syntactical information retrieval has to be incorporated into the sign recognition process from the very beginning. That is what makes it impossible to add a 'syntax module' later. That is why this is such a fundamental problem.
To really build a natural sign interpreter, which for me is the ultimate goal of sign language recognition research, one has to process spatial information. This means setting up a system that takes this spatial information into account in every phase.

5.4 Future research

Future research will have to find ways of detecting spatial/syntactical information and incorporating the gathering and interpreting of this kind of information into the sign recognition process. Other areas that need to be investigated are recognition of facial expressions and mouth shapes and better recognition of handshapes. Also, ways have to be found to make a user-independent system, either by generalising enough or by constructing a training mechanism with which a user can adapt a system to himself. Continuous sign recognition is also a field in which more research could be done, to find ways of dealing with co-articulation effects and movement epenthesis. It is also vital to perform more linguistic research on (Dutch) sign language, so that a theoretical base for model-building rules can be created. Linguistic research comparing different sign languages to find out which characteristics (if any) of signs and sign grammars are universal would also be extremely useful; not only for model building, but also for creating sign recognition methods which could be applicable to many or even all sign languages. And finally, it would be interesting to investigate the sign representation Crasborn (2003) suggests, and see whether this could be used in automatic sign language recognition.

References

Bahrami, Sh., Vannobel, J.M. (2001) A parallel and hierarchic framework to visual gesture recognition. Paper presented at the International Gesture Workshop (GW 2001), London, UK, April 18-20.

Bauer, B., Kraiss, K.-F. (2002) Video-Based Sign Recognition using Self-Organizing Subunits. In: Kasturi, R. , Laurendeau, D. , Suen, C. (eds.): *Proceedings of the 16th International Conference on Pattern Recognition (ICPR 2002), Québec City, Canada, August 11-15 (on CD-ROM)* , pp. none. IEEE Computer Society.

Bauer, B., Kraiss, K.-F. (2001) Towards an Automatic Sign Language Recognition System Using Subunits. Paper presented at the Gesture Workshop 2001, London, UK, April 18-20.

Bauer, B., Hienz, H. (2000) Relevant Features for Video-Based Continuous Sign Language Recognition. In: *Proceedings of the 4th International Conference on Automatic Face and Gesture Recognition (FG 2000), Grenoble, France, March 28-30*, pp. 440-445.

Bauer, B., Niessen, S., Hienz, H. (1999) Towards an Automatic Sign Language Translation System. In: Università degli Studi di Siena (ed.): *Towards New Paradigms for Interaction Beyond the Desktop: Proceedings of the 1st International Workshop on Physicality and Tangibility in Interaction, Siena, Italy, October 20-22.*

Bowden, R., Windridge, D., Kadir, T., Zisserman, A., Brady, M. (2004) A Linguistic Feature Vector for the Visual Interpretation of Sign Language. In: Pajdla, T., Matas, J. (eds) *Proceedings of the 8th European Conference on Computer Vision (ECCV'04), Prague, Czech Republic, May 11-14*, volume 1, pp. 391- 401, Springer-Verlag, Heidelberg.

Bowden, R., Zisserman, A., Kadir, T., Brady, M. (2003) Vision based Interpretation of Natural Sign Languages. Poster presented at the 3rd International Conference on Computer Vision Systems (ICVS'03), Graz, Austria, April 2003.

Bowden, R., Sarhadi, M. (2002) A non-linear model of shape and motion for tracking finger spelt American sign language. "Image and Vision Computing", **20**, 9-10, pp. 597-607.

Braffort, A. (2002) Research on Computer Science and Sign Language: Ethical Aspects. In: Wachsmuth, I., Sowa, T. (eds) *Gesture and Sign Language in Human-Computer Interaction: Proceedings of the International Gesture Workshop (GW 2001), London, UK, April 18-20*, pp. 1-8. Springer-Verlag, Heidelberg.

Chen, Y., Gao, W., Fang, G., Yang C., Wang, Z. (2003) CSLDS: Chinese Sign Language Dialog System. In: *Proceedings of the IEEE International Workshop on Analysis and Modelling of Faces and Gestures (AMFG'03), Nice, France, October 17*, pp. 236-238. IEEE Computer Society.

Crasborn, O. (2003) Cylinders, planes, lines and points: Suggestions for a new conception of the handshape parameter in sign languages. In: Cornips, L., Fikkert, P. (eds): *Linguistics in the Netherlands*. Benjamins, Amsterdam, pp. 25-32.

Dellaert, F. (2004) "Software". Found on June 6, 2004 at the WWW: http://www.cc.gatech.edu/~dellaert/html/software.htm.

Deuchar, M. (1984) *British Sign Language*. Routledge & Kegan Paul, London.

Emmorey, K., Falgier, B. (2004) Conceptual locations and pronominal reference in American Sign Language. "Journal of Psycholinguistic Research", **33**, 4, pp. 321-331.

Emmorey, K. (2002) *Language, Cognition and the Brain: Insights from Sign Language Research*. Lawrence Erlbaum Associates, Mahwah, NJ.

Fang, G., Gao, W., Zhao, D. (2003) Large Vocabulary Sign Language Recognition Based on Hierarchical Decision Trees. In: *Proceedings of the 5th international conference on Multimodal interfaces (ICMI'03), Vancouver, Canada, November 5-7*, pp. 125 - 131.

Fang, G., Gao, W., Chen, X., Wang, C., Ma, J. (2002) Signer-independent Continuous Sign Language Recognition Based on SRN/HMM. In: Wachsmuth, I., Sowa, T. (eds) *Gesture and Sign Language in Human-Computer Interaction: Proceedings of the International Gesture Workshop (GW 2001), London, UK, April 18-20*, pp. 76-85. Springer-Verlag, Heidelberg.

Gao, W., Ma, J., Wu, J., Wang, C. (2000) Sign Language Recognition Based on HMM/ANN/DP. "International Journal of Pattern Recognition and Artificial Intelligence", **14**, 5, pp. 587-602.

Groce, N.-E. (1985) *Everybody Here Spoke Sign Language; Hereditary Deafness on Martha's Vineyard.* Harvard University Press, Cambridge, Massachusetts.

Gurney, K. (1997) *An introduction to neural networks*. UCL Limited, London.

Holden, E.-J., Owens, R. (2003) Recognising Moving Hand Shapes. *Proceedings of the 12th IEEE International Conference on Image Analysis and Processing (ICIAP'03), Mantova, Italy, September 17-19*, pp. 14-19. IEEE Computer Society.

Holden, E.-J., Owens, R., Roy, G. (2001) Visual Sign Language Recognition. In: Klette, R., Huang, T., Gimen'garb, G. (eds) *Multi-Image Analysis: Proceedings of the 10th International Workshop on Theoretical Foundations of Computer Vision, Dagstuhl Castle, Germany, March 12-17, 2000*, pp. 270, Springer-Verlag, Heidelberg.

Holden, E.-J., Owens, R., Roy, G. (1999) Adaptive Fuzzy Expert System for Sign Recognition. *Proceedings of the International Conference on Signal and Image Processing (SIP'2000), Las Vegas, USA, November 19-23*, pp. 141-146.

Kadous, M.W. (1996) Machine recognition of Auslan signs using PowerGloves: Towards large-lexicon recognition of sign language. In: *Proceedings of the Workshop on the Integration of Gesture in Language and Speech (WIGLS'96), Delaware, USA, October 7-8*, pp. 165-174. Wilmington, DE.

Lejeune, F., Braffort, A., Desecles, J.-P. (2002) Study on Semantic Representations of French Sign Language Sentences. In: Wachsmuth, I., Sowa, T. (eds) *Gesture and Sign Language in Human-Computer Interaction: Proceedings of the International Gesture Workshop (GW 2001), London, UK, April 18-20*, pp. 197-201. Springer-Verlag, Heidelberg.

Nyst, V. (2004) Iconicity in Adamrobe Sign Language. Presentation at the Mini-conference on Sign Lanugage Research, Nijmegen, the Netherlands, July 13, 2004.

Oja, E. (2003, October 31) "HUT – CIS: Independent Component Analysis". Found on April 28, 2004 at the WWW: http://www.cis.hut.fi/projects/ica/

Ong, E., Bowden, R. (2004) Detection and Segmentation of hand shapes using Boosted Classifiers. In: *Proceedings of the 6th International Conference on Automatic Face and Gesture Recognition (FGR 2004), Seoul, Korea, May 17-19*, pp. 889-894.

Rabiner, L., Juang, B.-H. (1993) *Fundamentals of Speech Recognition*. Prentice Hall PTR, Englewood Cliffs, NJ.

Ross, T. (1995) *Fuzzy Logic with Engineering Applications*. McGraw-Hill., New York.

Russel, S., Norvig, P. (1995) *Artificial Intelligence, A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

Sallandre, M.-A., Cuxac, C. (2002) Iconicity in Sign Language : a theoretical and methodological point of view. In: Wachsmuth I., Sowa T. (eds.): *Gesture and Sign Language in Human-Computer Interaction: Proceedings of the International Gesture Workshop (GW 2001), London, UK, April 18-20*, pp. 171-180. Springer-Verlag, Heidelberg.

Starner, T., Weaver, J., Pentland, A. (1998) Real-time American Sign Language using desk and wearable computer based video. "IEEE Transactions on Pattern Analysis and Machine Intelligence", **20**, 12, pp.1371 -1375. IEEE Computer Society.

Vamplew, P., Adams, A. (1998) Recognition of sign language gestures using neural networks. "Australian Journal of Intelligent Information Processing Systems", **5**, 2, pp. 94-102.

Vogler, C., Metaxas, D. (2001) A Framework for Recognizing the Simultaneous Aspects of American Sign Language. "Computer Vision and Image Understanding", **81**, pp. 358-384.

Vogler, C., Metaxas, D. (1999a) Toward Scalability in ASL Recognition: Breaking Down Signs into Phonemes. In: Braffort, A., Gherbi, R., Gibet, S., Richardson, J., Teil, D. (eds) *Gesture-Based Communication in Human-Computer Interaction: Proceedings of the International Gesture Workshop (GW'99), Gif-sur-Yvette, France, March 17-19*, pp. 211-226. Springer-Verlag, Heidelberg.

Vogler, C., Metaxas, D. (1999b) Parallel Hidden Markov Models for American Sign Language Recognition. In: *Proceeding of the 7th IEEE International Conference on Computer Vision (ICCV'99), Kerkyra, Greece, September 20-27*, pp. 116-122. IEEE Computer Society.

Vogler, C., Metaxas, D. (1997) Adapting Hidden Markov Models for ASL Recognition by Using Three-dimensional Computer Vision Methods. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC'97), Orlando, Florida, USA, October 12-15*, pp. 156-161. IEEE Computer Society

Wang, C., Gao, W., Ma, J. (2002) A Real-Time Large Vocabulary Recognition System For Chinese Sign Language. In: Wachsmuth, I., Sowa, T. (eds) *Gesture and Sign Language in Human-Computer Interaction: Proceedings of the International Gesture Workshop (GW 2001), London, UK, April 18-20*, pp. 86-95. Springer-Verlag, Heidelberg.

Zeshan, U. (2004a) Posession in sign language: a typological pilot study from China, Turkey, India, Korea, Russia and Germany. Presentation at the Mini-conference on Sign Language Research, Nijmegen, the Netherlands, July 13, 2004.

Zeshan, U. (2004b) Interrogative constructions in signed languages: crosslinguistic perspectives. "Language", **80**, 1, pp. 7-39.

Appendix I: Parameter Categories of the Linguistic Feature Vector

Hand Arrangement (HA):

1. right hand high
2. left hand high
3. hands side by side
4. hands are in contact
5. hands are crossed

Position (TAB):

1. the neutral space
2. face
3. left side of face
4. right side of face
5. chin
6. right shoulder
7. left shoulder
8. chest
9. stomach
10. right hip
11. left hip
12. right elbow
13. left elbow

Movement (SIG):

1. hand makes no movement
2. hand moves up
3. hand moves down
4. hand moves left
5. hand moves right
6. hands move apart
7. hands move together
8. hands move in unison

Handshape (DEZ):

1. 5-hand
2. A-hand
3. B-hand
4. G-hand
5. H-hand
6. V-hand

Appendix I – Parameter Categories of the Linguistic Feature Vector

Vocabulary word:	English translation:
ik	I
werken	to work
kantoor	office
gaan	to go
vandaag	today
niet	not
ziek	ill
maar	but
morgen	tomorrow
vrijdag	Friday
als	if
beter	better/well
weekend	weekend
ons	our
gezellig	convivial
collega	colleague
wij	we
kletsen	to chat
veel	much/a lot
ook	also
1	one (1)
hij	he
oud	old
moeder	mother
man	man
hele	entire
dag	day
andere	other
praten	to talk
mogen	to like

Appendix III: Programming code

1. Read function

The read function removes garbage from a binary feature vector data file and returns the clean binary feature vectors. It receives three arguments: the file to be read, the number of the first frame to be included, and the number of the first frame not to be included any more. So `read('file1',12,24)` means read frame 12 up to and including frame 23 from file 'file1' and return the clean feature vectors. An example of cluttered and clean data is given in section 3.5.3 about data format.

```
#include "features.h"
using namespace std;

vector<vector<int> > readargs(char *filename,char *start,char *finish)
{
  string
    str, begin, end;
  vector<intvec>
    featurevecs;
  ifstream
    vecfile;

  vecfile.open(filename);      // open the sourcefile

  if(!vecfile)
    {
      cerr << "creating ifstream object failed\n";
      exit(1);
    }

  intvec          // make a new int-vector
    temp;
  bool
    go_on = true;

  // start- and end strings:
  begin = start;
  end = finish;
  bool flag = false;

  while(getline(vecfile,str))
    {
      if (str.find(end) != string::npos    // if you find 'end',
      break;                               // stop reading
      // as long as you haven't found 'begin', don't read
      if ((!flag) && (str.find(begin) == string::npos))     {
        continue;
      }
      flag = true;       // set the flag: now you know 'begin' was found

      if(!go_on)         // if you just added a vec, temp must
      temp.clear();      // be cleaned for the new vec

      if(str.find('|') != string::npos)
```

```
    {
      str.erase(0,str.find('|')+1);     // cut off garbage
          // put the figures in temp
      for(string::size_type i=0;i < str.length();i++)        {
          switch(str.at(i))
          {
          case('1'):
            temp.push_back(1);
            break;
          case('0'):
            temp.push_back(0);
            break;
          }
        }
      go_on = true;  // end of the vec was not reached
    } else
      go_on = false; // line without '|': end of vec is reached

    if( (!temp.empty()) && (!go_on) )    // if the end is reached and
                                         // temp is not empty..
    featurevecs.push_back(temp);  // .. add temp's contents at the
                                  // end of featurevecs
    str = "";                     // clear str

  }

  return featurevecs;   // return the result:an array of binary vectors
}
```

2. Lookup table function

The *lut* function creates a lookup table associating vectors with symbols. The `lut` function receives a matrix consisting of N row vectors as an argument, clusters the vectors and generates a symbol for each centroid vector. It returns an N x 2 cell matrix containing the centroid vectors and their newly generated associated symbols.

```
function lookup = lut(matrix,varargin)
% Creates a lookup table from transformed, clustered vecs.
% Lut(matrix) creates a lookup table from
% vectors to symbols.
% The vectors' resemblence can be measured in Euclidian distance
% Lut operates by calling kmeans to cluster the vectors, then
% storing all not-'NaN' cluster centroids in the lookup table.
% The output is the lookup table.
% The lookup table is a cell array:
% column 1 contains the vectors, column 2 the symbols associated
% with them. The symbols are numbers.
% varargin is an optional argument containing the desired number
% of clusters. Default is 10.

difvecs = [];
K = 10;     % Number of clusters; could be passed as arg, too.
if nargin>1 % If so, use that nofclusters.
    if isnumeric(varargin{1})
        K = varargin{1};
    else
```

```
        fprintf('\nWarning: nclus is not numeric; default of nclus = 10
is used.');
    end
end
[m,n] = size(matrix);
if (m==0) || (n==0)      % check size of matrix
    disp 'warning: matrix has zero dimensions!';
end

% Do kmeans. for this function, matrix must be transposed
[labels,centroids] = kmeans(matrix',K);
centroids = centroids'  % contains the clus centroids as row vecs.

% Place the valid vectors in the lut:
[x,y] = size(centroids);
lut = {};
j = 1;
for i = 1:x
    if isnan(centroids(i,1))
        continue
    end
    lut(j,1) = {centroids(i,:)};
    lut(j,2) = {j};
    j = j+1;
end
lookup = lut;
```

3. Symbol functions

3.1 Symbol

The `symbol` function receives a *lut* and a matrix, consisting of row vectors, as its arguments. First, the `symbol` function uses the `findsymbs` function to find the correct symbols for the vectors. `findsymbs` returns a symbol array. The `symbol` function receives this array and transforms it into a better format: a two-column cell matrix. The first column contains the symbol, the second the number of times this symbol occurs in a row. This format is more compact and easier to build models with than the simple output of `findsymbs`. The symbol table is a cell array rather than an ordinary one to facilitate the use of other data types as symbols, such as letters or strings: in a cell array, any type can be stored.

```
function symb = symbol(vectors,lut)
% Symbol turn a series of vectors into a series of symbols
% These symbols are looked up in the lut
% With this symbol series, a symbol table is created. The symbol table
% is a cell array with a symbol in the first row, and the number of
% appearences-in-a-row in the second

s = findsymbs(vectors,lut);
n = length(s);
if n == 0
    error('The number of symbols in the symbol series is 0')
end
symb = cell(1,2); % this is the cell array: symbol-number-of
                  % symbs
symb{1,1} = s(1);
symb{1,2} = 1;
```

```
j = 1;              % j is the index for the cell array 'symb'
for i = 2:n         % i is the index for the symbols array 's'
    if s(i) == s(i-1)                % if next symbol is equal..
        symb{j,2} = symb{j,2} +1;    % .. raise the number-of
    else
        j = j+1;                     % if it is not equal..
        symb{j,1} = s(i);            % .. raise j and record a new
                                     % symbol..
        symb{j,2} = 1;               % .. and give it number-of: 1
    end
end
```

3.2 Findsymbs

`findsymbs` receives a *lut* and a matrix of row vectors. For each vector, it checks the *lut* to find the *lut* vector that lies closest to it in Euclidian terms. The symbol associated with that closest *lut* vector is stored in an array. This way, findsymbs translates a series of vectors into a series of symbols. It returns the array of symbols.

```
function s = findsymbs(vectors, lut)
% Finds symbols for euclidian measurable vectors in a lookup table

[m,n] = size(vectors);
[x,y] = size(lut);
s = [];
for i = 1:m
    mindif = 10000;
    ind = 1;
    for j = 1:x
        dif = sum((vectors(i,:)-lut{j,1}).^2.^.5);
        if dif < mindif
            mindif = dif;
            ind = j;
        end
    end
    s = [s,lut{ind,2}];
end
```

4. Model functions

Model

The function `model` receives a lookup table (lut) and a variable number of vector matrices for which models must be made. It returns a cell array containing the Markov chain models in table form, one for each input matrix. A table model is itself a cell matrix, containing the state symbols in the first row and column, and the transition probabilities in the other cells. `Model` uses `model1` to create a sign model. It uses it to create one for each of its argument matrices, and concatenates the results into one cell array, which it returns.

```
function mods = model(lut,varargin)
% model creates models from the matrices with the aid of the lut.
% model is a noname version of the function. Names for the models
% do not have to be provided; they are looked up via the
% names function.

if nargin < 2
    error('No vector lists to make models from')
```

```
end
mods = {};
for i = 1:length(varargin)
    [x,y] = size(varargin{i});
    mat = varargin{i};
    mod = model1(mat,lut);
    mods{1,i} = mod;
end
```

Model1

`model1` receives a matrix of vectors and a lut. It uses `symbol` to extract the symbol table from the vector matrix with the aid of the lut. With this table, it is easy to build a model: the subfunction `maketrans` creates a new state (= row and column in the model table) for each symbol in the symbol table. Then it calculates the transition probabilities and enters them into the table. These are easily calculated from the number of occurrences of a symbol recorded in the symbol table. If a symbol occurs 3 times in a row, the transition probabilities are: autotransition: 0.67, next-transition: 0.33. Since no other transitions are allowed, calculating probabilities is no more complicated than that. `model1` returns the cell matrix model.

```
function mod = model1(matrix,lut)
% Model1 creates a Markov Chain model of a sign.
% Model1(matrix,lut) creates a Markov Chain model
% in the form of a matrix that represents
% the transition probabilities of the states.
% 'Matrix' is a series of row vectors. Each vector becomes a state.
% Each state has a transition to the next state, and
% an optional transition to itself. The probabilities of
% these transitions are represented in a matrix.
% This matrix is a cell array. The first line and column contain the
% symbols for the states. The other cells contain the transition
% probs from the state in the row to the state in the column
% of the matrix.
% So the trans prob of state 'X' to stat 'Y' is found in the cell
% whose row has 'X' in the first column and whose column has 'Y'
% in the first row.
% In effect, the cell array is a table, with symbols as row and column
% indices.

vecs = matrix;
[m,n] = size(vecs);      % there are m vectors with dim. n
[z,q] = size(lut);       % there are z unique vectors
if (m == 0) || (n == 0) || (q == 0) || (z == 0)
    error('There are dimensions in matrix or lookup that are 0');
end
if (q ~= 2) || (~iscell(lut))
    error('The lookup must be a cell array of 2 columns.');
end
[x,y] = size(lut{1,1}); % the lookup vecs have dim. y
if n ~= y
    error('Dimensions of vectors and lookup do not match. \nVector dim.
is %-1.0f, Lookup vector dim. is %-1.0f',n,y);
end

% This serie of vecs is a series of symbols. Which?
% Find the appointed symbols in the lut and record them in the form
% of a symbol table (a cell matrix) with the function symbol.
symbs = symbol(matrix,lut);
```

```
% Now create a table (a cell matrix again) in which the transition
% probabilities can be stored.
trans = maketrans(symbs);

mod = trans;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Subfunctions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function trans = maketrans(s)
% Creates the transition matrix from a symbol table.
% The result is a cell matrix. First row and column contain the
% symbols, the other cells contain the transition probabilities
% from row symbol to column symbol.
% So the cell {2,4} contains the transition prob from the state
% with the symbol in {2,1} to the state with the symbol stored
% in {1,4}
% S is the symbol table
% With this, the transition probs can be calculated easily.

[n,m] = size(s);
if (n == 0) || (~iscell(s)) || (m ~= 2)
    error('The symbol row has length 0 or is not a 2-column cell
      array.')
end
trans = {};
for i = 1:n-1
    trans{i+1,1} = s{i,1};     % Put the symbols in the 1st row & column
    trans{1,i+1} = s{i,1};

    num = s{i,2};         % number of times symbol i occurs in a row

    trans{i+1,i+1} = (num-1)/num ;  % num-1 of the num occurrences mean
                                       autotransition..
    trans{i+1,i+2} = 1/num;          % .. de remaining one is to the
                                         next symbol
end
trans{n+1,1} = s{n,1};          % treat the final state differently
trans{1,n+1} = s{n,1};
if s{n,2} == 1                  % its prob is always 1 or 0
    trans{n+1,n+1} = 0;
else
    trans{n+1,n+1} = 1;
end
```

5. Test functions

Sdotest

The function `sdotest` receives a lookup table (lut), a cell array of signs to be classified and a cell array of models as arguments. The cell array of signs contains the vector matrices of the new, unseen signs, transformed with the ICA unmixing matrix. These are the test signs. The cell array of models contains the transformed vector matrices of the example signs, two for each sign in the vocabulary.

`Sdotest` checks the probability of all models for each test sign with the aid of `stest`. It organises the results in on big cell matrix containing the test signs in the first column, and the probability lists of the models for each sign in the second. It then returns this cell matrix.

```
function prob = sdotest(lut,number,signs,varargin)
% Calculates the probabilities of each model in varargin for all signs
% lut is the lookup table.
% signs is a 1 x M cell array containing vector series
% varargin are the models to compare with; also vector series
% number indicates how many of each sign you are testing. This
% to facilitate the 'names' function.
% The output is a cell array sign - prob list for all models
if nargin < 4
    error('no models or no lut or no nofsigns given');
end
mods = {};
%prob = {};
[n,m] = size(signs);
if n ~= 1 || ~iscell(signs)
    error('signs must be given as a 1 x M cell array')
end

% Do an stest for each sign in signs and store the output in prob
for i = 1:m
    fprintf('\n');
    p = names(ceil(i/number))
    out = stest2(lut,signs{i},varargin)
    prob{i,1} = names(ceil(i/number));
    prob{i,2} = out;
end
```

Stest

`stest` receives a lut, a new sign and the model signs as arguments. It transforms the sign and the models into symbol series with the aid of `symbol`. Then it compares the symbol sequences using `scompare`. `stest` receives the comparison score for each model from `scompare`. It orders these according to score, with the highest score on top, and stores the first fifteen scores in the form of a two-column cell matrix. The first column contains the names of the models, the second column their scores.

```
function out = stest(lut,new,mods)
% Compares 'new' to each of the 'mods'.
% new is a new sign, in the form of vectors series.
% mods is a 1 x M cell array of models.
% lut is the lookup table in which to find the symbols for vectors.
% stest finds the symbol sequences for all models and does an scompare
% between the new sign and the models.
% The output is a list with in the first column the name of a model,
% and in the second the probability of that model producing the
% new sign.

if nargin<3
    error('No models to compare with or no lookup table');
end
if ~iscell(lut)
    error('lookup table must be a cell array');
end
```

```
list = [];

% Get the symbol sequence from the new matrix
new = symbol(new,lut);
new = spull(new);              % pulls the symbol sequence from a table

% Read the arguments and pull their symbol sequences, too
for i = 1:length(mods)
    tmp = symbol(mods{i},lut);
    tmp = spull(tmp);
    score = scompare(new,tmp);      % compare each model to the newsign
                                        % matrix
    list = [list;i/2,score];        % add model nr (=i/2, because
                                        % there are 2 mods per sign)
                                        % and score to the output
end
list = flipud(sortrows(list,2));    % put in order of highest prob.
if length(mods) > 15
    list = list(1:15,:);            % only output the best 15
end
out = {};                       % output as list of name - prob.
for i = 1:length(list)
    k = list(i,1);
    out{i,1} = names(round(k));
    out{i,2} = list(i,2);
end
```

Scompare

scompare compares two symbol tables and checks if their symbol sequences are identical. It receives two symbol sequences as arguments. If they are identical, it returns 1, if not, it returns 0.

```
function val = scompare(slist1,slist2)
% scompare compares two symbol lists.
% It checks whether the state sequences are equal.

val = 0;
if length(slist2) == 0
    error('The second symbol list has length zero.');
end
if length(slist1) == 0
    error('The first symbol list has length zero.');
end
if length(slist1) ~= length(slist2)
    return
end
if all(slist1 == slist2)
    val = 1;
end
```

Scalcp

scalcp calculates the success rate on the basis of a test result matrix. It receives the result cell matrix as its argument and returns the percentage of correctly classified signs. Since the model probability lists are

ordered with the highest on top by `stest`, `scalcp` simply checks whether the model name at the top of the list matches the name of the test sign. A loose criterion was chosen: if there was a tie, with more than one model having the same top probability, classification was counted as correct as long as one of the top models was the correct one.

```
function p = scalcp(ptable)
% calculates the percentage of correctly identified signs in ptable
% ptable is a cell array of signname - prob list of models
% only when the top probability model is indeed the model for sign
% signname, is the sign registered as correctly identified.
% The prob tables that ptable contains must be ordered in
% descending order with the top probs at the top.
% (they are delivered thus by function sdotest)

[m,n] = size(ptable);
if ~iscell(ptable) || n ~= 2
    error('ptable must be a M x 2 cell array');
end
score = 0;
for i = 1:m
    sign = ptable{i,1};
    probs = ptable{i,2};
    ts = probs{1,2};
    j = 1;
    while (probs{j,2} == ts) && (j <= length(probs))
        if length(probs{j,1}) == length(sign)
            if probs{j,1} == sign
                score = score+1;
                break
            end
        end
        j = j+1;
    end
end
p = score/m;
```

Scompare2

`scompare2` compares two symbol sets.

```
function val = scompare2(slist1,slist2)
% scompare compares two symbol lists.
% It counts how many of the symbols in slist2 do indeed occur
% in slist1 and divides by the number of symbols in slist2.
% It also calculates the difference in length between both lists
% and divides by the length of slist2.
% It then averages these two percentages to calculate the final score.

val = 0;
if length(slist2) == 0
    error('The second symbol list has length zero.');
end
if length(slist1) == 0
    fprintf('\nWarning: first symbol list has length zero.');
end
for i = 1:length(slist2)
```

```
    f = find(slist1 == slist2(i));
    if ~isempty(f)
        val = val+1;
    end
end
val = val/length(slist2);
dif = (length(slist1)-length(slist2)) ^2^.5;
if dif >0
    val = (val + 1/(dif/length(slist2)) )/2;
end
```

6. Model names function

names looks up the name of a sign with the aid of its code number.

```
function name = names(n)
% Looks up the sign name associated with the sign number
switch n
    case 1
        name = 'ik';
    case 2
        name = 'werken';
    case 3
        name = 'kantoor';
    case 4
        name = 'gaan';
    case 5
        name = 'vandaag';
    case 6
        name = 'niet';
    case 7
        name = 'ziek';
    case 8
        name = 'maar';
    case 9
        name = 'morgen';
    case 10
        name = 'vrijdag';
    case 11
        name = 'als';
    case 12
        name = 'beter';
    case 13
        name = 'weekend';
    case 14
        name = 'ons';
    case 15
        name = 'gezellig';
    case 16
        name = 'collega';
    case 17
        name = 'wij';
    case 18
        name = 'kletsen';
    case 19
        name = 'veel';
```

```
    case 20
        name = 'ook';
    case 21
        name = 'een';
    case 22
        name = 'hij';
    case 23
        name = 'oud';
    case 24
        name = 'moeder';
    case 25
        name = 'man';
    case 26
        name = 'hele';
    case 27
        name = 'dag';
    case 28
        name = 'andere';
    case 29
        name = 'praten';
    case 30
        name = 'mogen';
    case 31
        name = 'daarom';
end
```