Gesture Recognition using Neural Networks

Anders Sandberg
nv91-asa@nada.kth.se

3rd June 1997

Abstract

Gesture interaction is a versatile, intuitive way of interacting with computers, especially suited for virtual environments and applications where many degrees of freedom are important. Human gestures have many meanings and uses, and are often different from person to person, while gesture input devices often introduce artifacts; this makes recognizing and interpreting gestures nontrivial. Gestures can be divided into postures, where the configuration of the hand is static, and true gestures, where the flexion of the fingers and the hand position/direction are changing dynamically. A recognition model based on a hybrid artificial neural network combining a radial basis function and a Bayesian classifier network is developed and tested with various forms of pre-processing or input representation. The results suggest that dynamic gesture recognition is feasible even for complex gestures, but that context information may be needed for reliable recognition.

Keywords: gesture recognition, artificial neural networks, radial basis functions, competitive learning, Bayesian neural networks, context sensitivity

Gestigenkänning med neuronnät

Referat

Gestinteraktion är ett flexibelt och intuitivt sätt att interagera med datorer, främst inom virtuella omgivningar och i tillämpningar där många frihetsgrader används. Mänskliga gester har många betydelser och användningsområden och är ofta olika från person till person, medan inmatningsenheter för gestinteraktion ofta introducerar artefakter; detta gör igenkänning och tolkning av gester problematisk. Gester kan delas upp i poser, där handens konfiguration är statisk, och verkliga gester, där fingarnas böjning och handens position och riktning förändras dynamiskt. En modell för igenkänning baserad på ett artificiellt hybridneuronnät som kombinerar radialbasfunktioner och ett Bayesianskt klassificeringsnätverk utvecklas och testas med olika former av förbehandling och indatarepresentation. Resultaten antyder att dynamisk gestigenkänning är genomförbart även för komplexa gester, men kontextinformation kan behövas för pålitlig igenkänning.

Nyckelord: gestigenkänning, artificiella neuronnät, radialbasfunktioner, competitiv inlärning, Bayesianska neuronnät, kontextkänslighet.

Preface

Gesture interaction is a very diverse field, where many applications and methods are related more by the fact that they are said to deal with gesture interaction than any inherent similarity. In this Master's thesis I have been looking mainly at the problems and possibilities of gesture interaction using a dataglove for virtual reality applications, and the use of a combined radial-basis function and Bayesian neural network for gesture recognition.

The main goals of this work has been to study the possibility of recognizing dynamic gestures in a flexible way with relatively few false positives and good generalization ability between different users. The number of gestures that have to be recognized in a virtual reality task is limited compared to sign language applications.

Acknowledgments

My thanks go to my supervisors Anders Lansner, Kai-Mikael Jää-Aro and Roland Orre for their support, guidance and suggestions.

I would also like to thank my friends who gave useful or interesting suggestions, and made gestures for use in the training: Johannes Keukelaar, Reine Bergström, Tobias Öbrink, Robert Söderberg, Henrik Öhrström, and Tommy Sääf.

My grandmother Ingrid Sandberg gave me a generous gift so that I could concentrate on my studies, for which I am very grateful.

Finally I owe many thanks to my mother, Birgit Sandberg, who forced me to work harder.

Contents

Chapter 1

Introduction to Gesture Interaction

Gesture interaction is a broad subject, involving aspects of psychology, linguistics, device design, human-computer interaction and pattern recognition. In this chapter I will discuss the why, what and how of gesture interaction.

1.1 Why Gesture Interaction?

One common goal of human-computer interaction is direct manipulation [Shn82], that the user feels that he/she interacts with the objects of the system directly instead of through an intermediary interface. Direct manipulation reduces cognitive load and makes building a mental model of the system easier for the user. But a major problem that remains is the ability to interact with the system in a useful manner; keyboard control is far too abstract for most interactive applications, mouse control is inherently 2-dimensional and voice control is still fairly limited outside narrow subjects if reliable function is required. Many applications require many degrees of freedom, but to control all of them using (say) joysticks, dials or mice is a daunting task for most people and require much training [Stu92].

This is especially problematic for immersive applications or where a high degree of intuitiveness is required. In an immersive application the user cannot see the keyboard or other interaction tools unless they are given a virtual representation, which is often an unsatisfactory solution[1] Another problem is that applications like virtual or enhanced reality are intended to be as natural and free-form for the user as possible, making it undesirable to use input devices that require previous training or a high level of symbolism. The ideal interaction method would be intuitive, explorative and direct.

[1] It is hard to implement a usable virtual keyboard resembling a normal keyboard since the accuracy of position trackers and finger-flexion measurements is too low to allow small keys, and a large virtual keyboard is too cumbersome to use. However, there exists chord-keyboards that could be used in virtual applications or with gesture input, such as the FingeRing of Fukumoto and Suenaga [FS94] (Kai-Mikael Jää-Aro, personal communication).

Gesture interaction appears to be a promising interaction paradigm for such applications. Gestures are an important part of daily human communication, and easily understood at least by other humans. While their exact execution varies, their meaning is more or less fixed, at least within the culture of the originator. In the same way the human hands are the main organs used for manipulation of our environment. The goal for virtual reality is to provide an interface to a virtual environment similar to reality in some aspects, and this makes gesture interaction attractive. It is hoped that by using gestures, the user can transfer skills into the virtual environment and the learning effort required to master the interface could be lessened [VB93].

1.2 Uses

Gesture interaction has been used in a variety of interfaces, with varying success and naturalness.

- Virtual reality has been one of the most popular applications. Typically a glove is mapped to a virtual hand used to manipulate the world and give commands to the system. The interaction model used in the Virtual Environment Display System of the Aerospace Human Factors Research Division of the NASA Ames Research Center and the VPL system has been widely used [Stu92]. Often gestures are used as a command language: a gesture is made to signal that the user wants the system to do something, such as move to another position or manipulate an object [JFH94].

- The Glove-Talk II system [Fel94] translates hand gestures into speech formants through an adaptive neural network interface. The gestures are mapped into parameters for a speech synthesizer, making it possible to "speak" in real-time using gestures.

- Obviously sign languages demonstrate the expressive abilities of gestures, although commonly only hand gestures are used in computer interfaces despite that movements of the body and facial expressions form a crucial part of most sign languages; this tends to limit the applications to finger spelling. Using images from a camera, ASL sentences can be recognized [SP90]. ASL letter and number recognition has also been achieved using a data-glove by Gregory Newby [New94] while Murakami and Taguchi [MT91] studied recognition of Japanese sign language.

- Multimodal interfaces involve the use of many different natural forms of input: speech, gesture and gaze. Especially the combination of speech and gesture has been studied much. It has been found that speech recognition can be improved significantly by lip-reading (which can be seen as a special form of gesture recognition) [BMHW93]. It is quite natural to use gestures for 3D design, for example in sculpting or curve specification [Wat93]. Weimer and Gannapathy used a dataglove combined with speech commands to specify b-spline curves [WG87].

- The CHARADE system [BBL93] uses hand gesture input to control a computer displaying slides during a presentation. This allows the user to go forward and backward in the presentation in various ways, for example move to the next chapter.

- Data-gloves have been used to control robots, both directly through a mapping from the human hand to an artificial humanoid hand [PS89] and using dynamic gestures [Bro89].
- Other possible applications may be puppetry [Stu92], control of a wearable computer [Man97], affective computing[2] [Pic95] or direct manipulation of 3D models.

1.3 Recognition vs. Interpretation

Gesture interaction essentially has two levels; gesture recognition, which identifies *what* the user is doing, and gesture interpretation, which identifies *why* the user is doing it. The first level is obviously the simplest, since it doesn't have to model the user's intentions or their relation to the context.

As Richard Watson points out [Wat93], pattern recognition consists of two subproblems: pattern representation and decision making. The raw data is processed by a representational subsystem into a form most convenient for the decision subsystem, which outputs the pattern classification (if any).

Context could help gesture recognition (if the user is making a movement which is similar to grabbing in the vicinity of a movable object, it is more probable that she wants to grab it than other alternatives), but is essential for gesture interpretation.

I have concentrated on the recognition (i.e. the simple) part in this work. However, I think neural networks could fill an important role in gesture interpretation, too. One possible architecture is having a low-level gesture recognizer network sending its output to a higher level network (or several) which takes into account both the recognized gesture, the actual data of the hand and context information from the rest of the systems (for example, hand distances to certain kinds of objects like doors, grabbable objects or tools could be encoded as activity of a group of input neurons). This higher level interpretation, which translates a situation into an actual command (open the door, grasp object or use a tool) could then send this command to the rest of the system. A hierarchy of levels could be used, merging various modalities in a way not unlike the way the human associative cortices are thought to process information in the brain. For a possible example of such a dynamic system, see [BMHW93] where acoustic and visual information are combined in a multilayer TDNN network.

1.4 Gesture as a Language

It should be noted that there exists many similarities between gesture and natural language recognition and interpretation [Wat93]. While gestures are

[2] Affective computing has been defined as computing that relates to, arises from, or deliberately influences, emotions.

not equivalent to language (if we exclude sign languages), gestures and speech complement each other [McN92], and gestures serve to communicate with the environment in a dynamic and highly interactive way. This produces similar problems in the two fields such as segmentation, differing speeds and users and especially interpretation/understanding.

Gestures have three major functions (Joëlle Coutaz, in [Wex95]):

Semiotic: Gestures used to communicate meaningful information, like signs or pointing gestures.

Ergotic: Gestures used to manipulate the world (real or virtual), like moving or transforming objects.

Epistemic: Gestures used in learning from the environment, like touching and testing objects.

In my work I will concentrate on the semiotic function, although I will also touch on the other two functions. Classically in gesture recognition the semiotic function has been studied the most, although the goal of direct manipulation involves finding ways of handling the ergotic function (as in the GIVEN system [BHV92] or the AHIG system described by Wexelblat [Wex94]). Often semiotic gestures like abstract grabbing or pointing gestures have been given an ergotic meaning, which introduces an unwanted level of interpretation, or ergotic gestures have been abstracted into semiotic gestures.

Sturman [Stu92] divides interpretation of hand sensor data into three categories: direct interpretation, mapped interpretation and symbolic interpretation (a slight change in terminology from the original publication in [SZP89]).

Direct interpretation is the most common category in immersive applications. Using the dataglove [Stu92], the user interacts with the application as if the application consists of physical objects manipulated in the real world. This form of interaction is obviously suitable for ergotic or epistemic functions.

In mapped interpretation the data from the hand is mapped onto virtual input devices such as the GKS standard devices [HDGS83]. The fingers may function as valuators, controlling a simulated vocal tract [Fel94], a simulated crane or robot [Stu92] through a linear or non-linear transfer function. In the Evhan system [Keu95] the simple gesture recognizer allows the user to fly with different speeds depending on the flexion of the thumb, which is a combination of mapped and symbolic interpretation.

Symbolic interpretation interprets gestures as tokens in a language. This ranges from simple syntax-free lookup in a lexicon over abstract grammars [HSM94] to a context sensitive syntax where the meaning of a gesture is determined from knowledge within the system. It is naturally suitable for semiotic functions.

I have concentrated my work mainly on symbolic interpretation, but it is compatible with mapped interpretation; by doing a symbolic gesture the hand can be turned into a mapped tool (for example a simulated mouse where hand position is used to move an object) and back again, or a postprocessor could use the data from the glove in combination with the gesture token calculated by my postprocessor to form a higher-level construct (this is similar to the system discussed by Alan Wexelblat in [Wex94]).

In symbolic interpretation one of the main issues is the lexicon of recognized gestures. What gestures it contains depends of course on the application, but also on what gestures can be recognized. As we will see, some recognition methods cannot adequately handle more than 10 gestures, which severely limits the repertoire of the user. On the other hand, a system using many gestures will be cumbersome to learn, unless real life experience can be brought into the system (as in the case of sign language).

1.5 When to Use Gestures and Not

It is worth noting that gestures are not always optimal for all applications. They are natural for some tasks (such as manipulating an object in a virtual environment), but unsuitable for other tasks (such as entering text). They also have some drawbacks: continuous use causes hand and arm tiredness, exact manipulations in space are difficult, the lack of force feedback makes the gestures feel "unreal", the large number of degrees of freedom of an input device can hinder precise manipulation since the number of possible directions to move in increases, and many of these DOFs are extraneous for the task at hand [JA96].

Many applications do not use gesture to their fullest, or use idiosyncratic choices of gestures that are perfectly obvious to the developer but does not fit well into the mental models of the users or even their culture. Good gestures are natural, which suggests using ergotic gestures rather than semiotic gestures, which are culture-bound [Mul96].

There doesn't exist any true standard for gesture interaction, but two gestures from the NASA system have become de facto standard in many virtual environment. These include flying (by pointing with the index finger in the right direction, regulating speed by flexing the thumb) and making a fist inside an object to grasp it. As [JA96] points out:

- They are reasonably intuitive and mimic real-life gestures
- They are actual gestures since the movement of the hand is relevant to regulate the direction of flying and where to move the object
- They are "spring-loaded" and only act when the user makes a conscious effort
- They benefit from being easily performed and sustained.

It is likely that gestures with these properties would improve the usability of gesture interaction in a system.

One way to increase usability and practicality of implementation is to adopt a design method. Sturman has proposed one such design method [Stu92], where evaluating the appropriateness of gesture interaction is the first step. He proposes answering questions in four main areas:

Naturalness: Are the tasks helped by pre-acquired sensorimotor skills, existing hand signs, absence of an intermediary device and that task control maps to hand actions?

Adaptability: Are diverse modes of control used in the tasks? Is it important to be able to switch modes of control rapidly and smoothly?

Coordination: Does the task require coordination of many degrees of freedom?

Real-time control: Are continuous monitoring and rapid user response important?

The answers to these questions suggests whether gestures are an appropriate interaction method for a certain application.

1.6 Gesture Input Devices

While we gesticulate with the entire body, gesture input has concentrated on hand and arm gestures. It should be noted that this loses many important gesture channels such as facial expression and body/torso posture, which are important in most social uses of gestures and a vital part of sign language.

The currently most used method for gesture interaction is the dataglove or datasuit, a glove or suit with sensors reacting to body movement and position. This is unfortunately somewhat cumbersome, since the users will be linked to the computer with wires or cables which limit free movements and often make them uncomfortable.

One possible improvement of the glove paradigm would be the use of wireless transmission of information, as has been envisioned for ubiquitous computing or wearable systems [Zim96] and in body tracking using electric fields [Smi96]. Especially visual methods such as scanning images from cameras and deducing hand/body position have been studied, although there remain many vision recognition, tracking and hardware problems to be solved. The earliest systems used infrared LEDs affixed to the person and viewed by several cameras, which deduced the 3D position of the LEDs. This was used by Poizner et al. for ASL recognition [PKBL83], although it was not done in realtime. One of the main problems with this kind of system was occlusions: the Videoplace system [Kru91] used silhouette tracking against a wall (and the later VIDEODESK against a desk [Kru93]) to circumvent the problem of self-shadowing. This system also allowed users to interact without special clothing or affixed devices, a desirable goal. Current research include gesture recognition using neural networks [Ban93] and Hidden Markov Models (HMMs) [SP90].

While there exists many different kinds of datagloves, such as mechanical devices like the Dextrous Hand Master of Exos Inc. (an exoskeleton affixed to the hand with Hall-effect sensors as potentiometers in the joints) or gloves using bending and stretching sensors such as the Digital Data Entry Glove [Gri83] and Nintendo PowerGlove[3] the most common type uses optical fibers. The "Sayre"

[3]The PowerGlove, manufactured by Mattel for the Nintendo games consoles, is worth mentioning as an extreme low cost dataglove, \$20 (compare this to the Exos DHM, \$15,000

Glove, one of the earliest such devices, used flexible tubes with a light source at one end and a photo-cell at the other. Finger flexion was measured by the extinction of light [DS77].

The VPL dataglove (Figure 1.1), which is one of the most commonly used gloves and the input device in this work, consists of ten optic fibers attached to a lycra glove with a Polhemus tracker[4] added on the dorsal side of the hand [ZLB+87]. The optic fibers measure bending angles by detecting the amount of light lost into the surrounding cladding, which is a function of the radius of curvature. It is thought [Kal93, p. 192] that the cables of the VPL glove have had small scratches inscribed at the joints, increasing the light loss when bent. The amount of loss seems to be an exponential function of bending angle (cf. [Kal93] for a more thorough treatment of how the VPL DataGlove works).

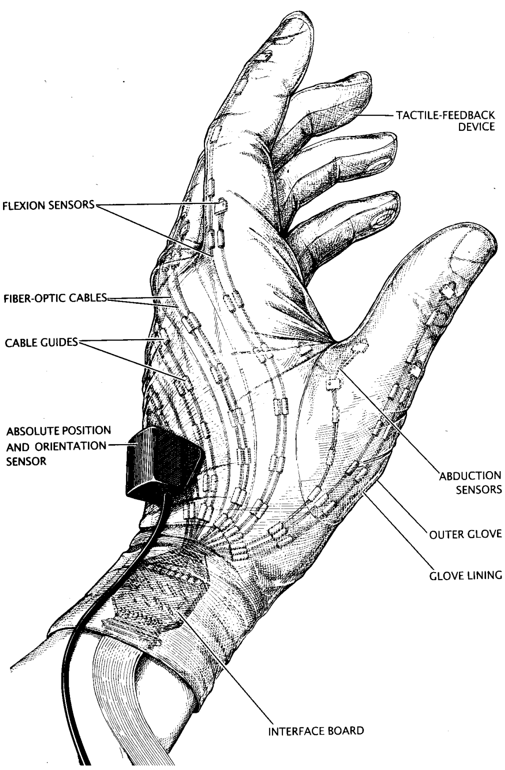


Figure 1.1: The VPL dataglove (from Scientific American, drawing by Carol Donner)

The glove measures both the metacarpophalangeal or MCP (the innermost) joints and the proximal interphalangeal or PIP joints (the outer joint of the

and the VPL DataGlove, \$8000). It uses resistive ink sensors, one for each finger, and has a comparatively low flexion resolution (two bits per sensor). Location is detected through ultrasonic sensors [Wat93].

[4] A three-space sensor made by Polhemus Inc. using low-frequency magnetic fields to sense orientation and position relative a source transmitter.

thumb and the middle joints of the other four fingers) using this method, producing ten degrees of freedom (with an additional six degrees of freedom from the Polhemus sensor).

The glove does not sense abduction, which means that the Star Trek "live long and prosper" V-sign will be interpreted as a slight bending of the joints instead of a spreading of the hand, although abduction sensors are available for the VPL glove. Some other gloves, like the Virtual Technologies Cyberglove [Wex94] also measure abduction and thumb rotation.

The VPL dataglove (and most other gloves) require calibration before use, since different people have differently sized hands (it comes in three sizes, and VPL recommended that users should choose the right size before use) and the glove characteristics may change due to wear or how it is worn on the hand. By making a series of standardized gestures (completely flat hand, flat hand with thumb bent and fingers rolled together) the values of the glove can be normalized to a standard range.

1.7 Static vs. Dynamic Gesture Recognition

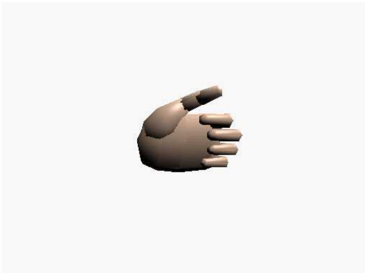
In the following I will distinguish between gestures and postures. A gesture is dynamic, while a posture is static (possibly a part of a gesture) [Mul96].

Gesture recognition is essentially a map between the joint angles of the fingers, the position and direction of the hand and possibly their time evolution to a set of gesture symbols (which could be parameterized, such as the NASA family of "fly" commands distinguished by a speed parameter or pointing gestures in various directions).

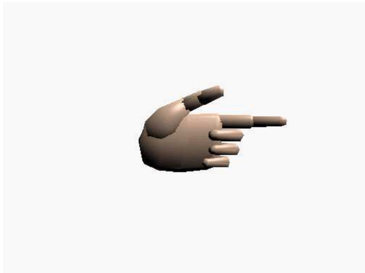
Mapping the signals from the glove device to a gesture can be done in many ways, especially since the information tends to contain redundancies (it requires some training to learn to move all fingers independently, usually the long-, ring- and little fingers tend to move as an unit).

The mapping can be based on only the latest information about the shape of the hand and fingers, its posture, or on its movement and past positions, which is called a gesture. Naturally postures are much easier to handle than the more general gestures; they are easy to learn and often correspond to simple metaphors, but they often cause tiredness, frustration (it is often hard to make a correct posture) and lack feedback [BBS94]. Many virtual environment systems have used standardized postures which are hard to form (cf. the common NASA postures [Bry93, p 1.3.24] in Figure 1.2).

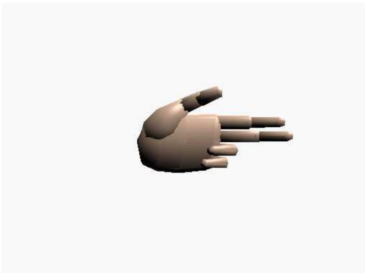
Gestures on the other hand involve movements, and are dynamic. The information is present not just in the positions, but how they change. It is assumed that the naturalness and intuitiveness is increased [JFH94, Kal93, BBS94, VB93], and they can more easily be placed in a context which limits the number of gestures that have to be distinguished at a given time [ZLB$^+$87]. On the downside, the amount of data that has to be processed increases, recognition becomes more difficult and some new problems appear like the speed of updates and especially the question of when a gesture actually begins.



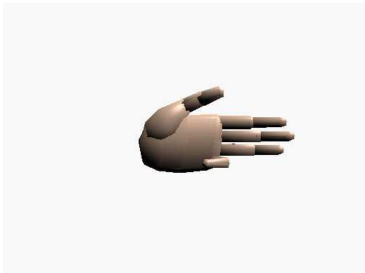
(a) Fist



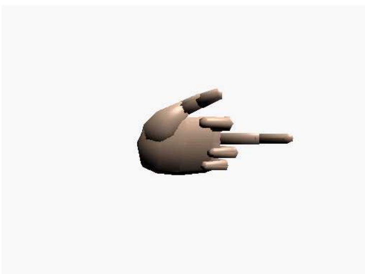
(b) Singlepoint



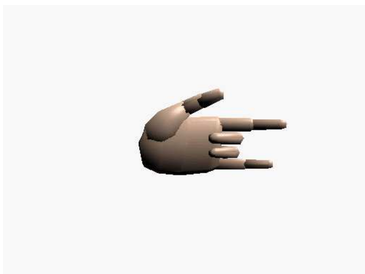
(c) Doublepoint



(d) Triplepoint

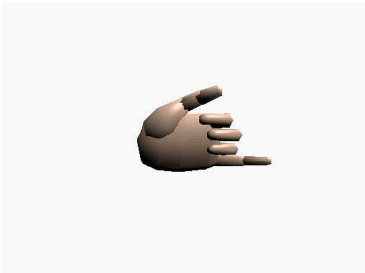


(e) Middlepoint

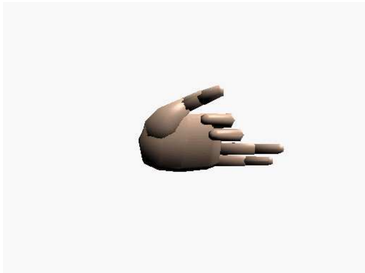


(f) Horns

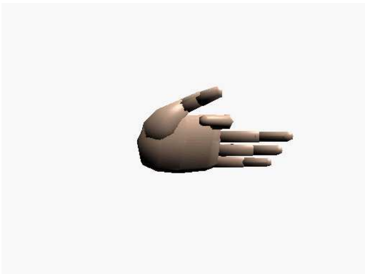
Figure 1.2: The gestures defined by NASA Ames Research Laboratories, and their designations. Note that the thumb is not significant in the gestures. Reprinted with permission from [JA96] (*Continued on next page.*)



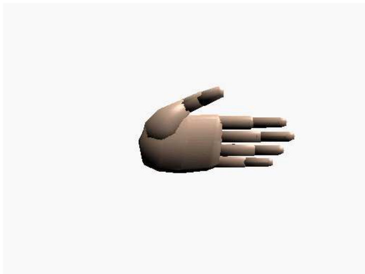
(g) Feathers



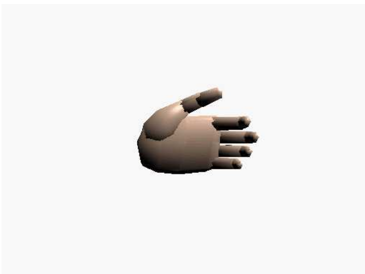
(h) Twofeathers



(i) Threefeathers



(j) Flat



(k) Null

Figure 1.2: (*Continued from previous page.*)

1.8 Problems in Gesture Recognition

There are several problems that complicate gesture recognition. They can be divided into hardware problems such as noise, device nonlinearities, lack of computing power, user idiosyncracies and variability, and fundamental problems such as the segmentation problem and context dependency.

1.8.1 Noise and Device Dependencies in the DataGlove

As in all physical systems, there is measurement noise in the input from a dataglove. In addition there is natural noise from the hand, which while extremely dextrous and steady in some respects also tends to vibrate, move slightly or slip within the glove. The sampling process also introduces some noise, theoretically around 0.4% for quantizing the data into 256 levels. I found the total noise level roughly equal to 2% when the hand wasn't moving and was held flat.

Wexelblat and Sturman [Wex94, Stu92] suggest that a low-pass filter should be used to reduce noise and spikes. This can be done using a linear causal IIR (Infinite Impulse Response) filter (after Sturman):

$$y_{i+1} = kx_i + (k-1)y_i, k > 0.5$$

where y_i are the filtered values and x_i the measured values. This will make responses a bit more sluggish (the characteristic time for this kind of filter is $\tau = -\frac{1}{\ln(1-k)}$).

Beside this noise, there is also the matter of device dependencies and crosstalk. Different kinds of input devices have different sensitivities and nonlinearities. In the VPL dataglove, the first nonlinear effect is the physical glove itself, which moves and flexes in a complex manner hopefully closely linked to the movements of the hand (it appears to follow the hand well, at least the parts containing the optical fibers, since most of the wrinkles which appear do so on the ventral side of the hand). The second nonlinearity is the conversion of bending angle into light reduction in the fibers and then A/D conversion into a number between 0–255[5], which appears to be a roughly exponential [Kal93] function of angle. Finally, calibration and various filters introduce a third nonlinearity. Sturman claims that this nonlinear response has little effect in practice, and hopefully it will be easy for the neural networks used in this work to handle. This has been supported by my experiments.

According to Sturman [Stu92] and personal experience, the finger-flex accuracy is less than the promised 1°, on the order of 5°.

The glove can transmit data at 60 or 30 samples per second over a 19.2 kbps connection with a computer, but at the higher rate only 32 bytes of data per record can be sent [VPL89], which is enough for the Polhemus location and direction and 10 bytes of flexion data from the MCPs and PIPs; it is also possible to poll the device for records when needed. Usually the real data rate is lower than this level, especially since the Polhemus sensor is often run at 20 Hz update frequency and the software introduces additional lag. A realistic speed is around

[5] Although the actual range is smaller, around 62–97

15 Hz, due to lags in the software and serial transmission. At this low rate of updating fast gestures ($\approx$ 3 m/s) or with quick changes in motion become problematic due to temporal aliasing [JA96].

Since the sensor for PIP bending has to pass the MCP joint, the flexion values will be somewhat affected by bending of the MCP[6]. This cross-talk is problematic for applications that require an accurate representation of the flexion angles, such as ergotic actions, but in many gesture applications the semiotic aspect is stronger and exact flexions are less important. Hong and Tan [HT89] have developed a method to calibrate the VPL glove by fixing one sensor to a known angle, systematically varying the other sensors and recording the readings. After this the known sensor angle was changed and the process repeated. This calibration was iterated over all sensors, and a function that removes the correlations could be derived.

These noise sources and nonlinearities suggests that the gesture recognition method must be robust, insensitive to noise and distortions. This points strongly in favor of adaptive methods, such as neural networks.

1.8.2 Computational Requirements

Since gesture interaction is in realtime, the recognition and interpretation methods have realtime constraints. If the system delays are too long the user will lose the direct interaction experience, and gestures will become largely symbolic (in addition to user irritation and confusion).

Fortunately the time constraints are fairly lax; an ordinary gesture takes between 1/10 and one second. Ideally the response should occur as soon as the user starts to make it, but this would increase the risk for misrecognitions (the system is too eager to assign meaning to any movement) and increase the computing load. A more reasonable recognition time would be a few tenths of a second for postures and perhaps even longer for complex gestures. Ergotic gestures of course require a much faster update rate than symbolic gestures, ideally corresponding to the update rate of the application.

These requirements place some limits on the complexity of the recognition methods, especially large neural networks.

Another aspect of this problem is communications: often the inter-process communications of a computer suffer noticeable lags due to inefficiencies in the operating system, which can decrease the performance.

1.8.3 Different Users, Different Speeds

Another problem is the differences between different people wearing the glove. A person with a small hand will wear it differently than a person with a large hand, and the calibrations are bound to be different from each other. In addition different people will usually make gestures and postures in different ways. To

[6] The correlation between the MCP and PIP in my experiments lies around 0.89 for the little, ring and middle fingers, 0.81 for the index finger and 0.46 for the thumb. How much of this that is due to bending interference and how much that is due to the fact that usually both joints are flexed together is uncertain.

further complicate the matter, even the same person will most likely wear the glove slightly differently on different occasions, and there might be a weak effect of how the optic fiber cable connecting the glove to the system is placed or moved during use.

Another severe problem is that users will do gestures with different speeds, especially as a response to sudden events or emotions. If a test subject is suddenly requested to make a gesture, it is often faster and shallower, especially if the subject is stressed.

This variance in speed has traditionally been the bane of many time-dependent neural networks like TDNN architectures, and is not easy to handle with many feature-detecting algorithms. One solution is to ignore the speed completely and re-scale time so that the posture evolves parameterized by path length [WB95]. Another is to "replay" the recent data at different magnifications [BBS94], time-warp (where time is stretched out or compressed depending on changes) or try to smooth out the signal between input nodes in a neural network.

1.8.4 The Segmentation Problem

One of the most complicated problems with recognition in general and gesture recognition in particular is to tell when a gesture starts and ends, and to identify its various parts. A gesture has in general no explicit starting and ending points, and cannot be repeated reliably. This is known as the segmentation problem.

The approach used by Wexelblat [Wex94] is to continuously monitor the input data stream, and report feature based descriptions that are interpreted by higher level systems.

But the segmentation problem is not just a question of determining what constitutes a gesture in the data, but also what constitutes a gesture to the user. When should a gripping motion trigger the appropriate response? Some users want the response to occur when the hand has closed completely into a fist, others want a quick reaction when they start the gesture, but this increases the likelihood of unintended reactions. Striking a balance between quick responses (and false positives) and secure responses (and frustration due to slowness) is hard and probably best seen as dependent on the application. If gestures have little effect or errors can be quickly undone (possibly relying heavily on ergotic gestures), then quick responses become advantageous, while in an application where a gesture has far-reaching consequences speed must be sacrificed for reliability. One solution would be to let the user control the sensitivity of the interface, and as their experience grows make it as sensitive they wish.

But the segmentation problem also complicates generating training data for the gesture recognizer (if it is trainable). The data has to be manually marked so that the recognizer knows what to signal, and this is equivalent to deciding when a gesture starts. A subtle source of errors here is the person doing the marking, who might be different from the intended user and will likely have different views of what constitutes an acceptable gesture and when it should trigger a response.

One way around the segmentation problem is to leave it to the gesture interpretation system. The recognizer simply sends its current confidence levels in different possible gestures to the interpreter, which has to decide when a gesture should trigger a reaction. This way data can be integrated over time (removing spurious patterns) and context can be considered.

1.8.5 Context Dependency

Gestures can be interpreted in different ways in different contexts; a pointing gesture towards a virtual object in a virtual environment means something different than a pointing gesture into empty space, and a twisting gesture could mean "close this valve", "open this door" or "turn this object" depending on where it is done and what objects are present. Obviously the context has to be taken into account when interpreting the gesture.

Some gestures are more meaningful or likely in one context than in another, which can aid recognition and interpretation. In the recognition step the likelihood estimate of the gesture classes can be improved by using context data, for example by calculating their conditional probability given the context.

One common method of providing context for gestures are tools, e.g. a pair of scissors help the ergotic cutting movement and a stick the epistemic poking movement [Mul96]. By holding a tool in the hand in a virtual environment, the repertoire of possible gestures will change and possibly their meaning. Tools are an intuitive metaphor, and have been used in various applications such as the Tankathon process-plant simulation of [JA92] to change the way the user interacts with the world. Unlike environmental context tool contexts are more clearly defined (either you hold a wrench in your virtual hand or not), although the interaction between a tool context and an environment context can become complex.

Context sensitivity appears to be a largely unexplored area, although there is much talk about how useful it could be [JA96, p. 14].

1.9 Gesture Recognition Methods

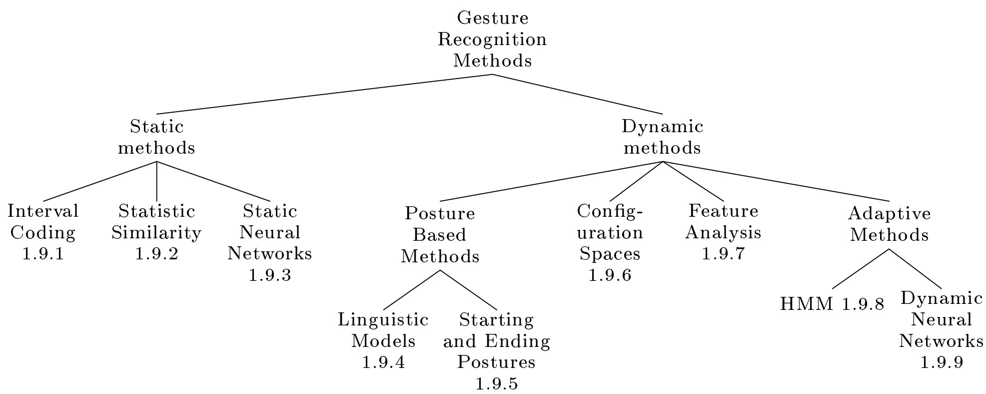


Figure 1.3: A partial taxonomy of gesture recognition methods. The numbers refer to the sections discussing the various methods.

Roughly speaking, gesture recognition can be divided into static and dynamic recognition methods (suitable for postures and gestures respectively), but dynamic recognition often builds upon static recognition, combining information of previous postures into a complex gesture.

1.9.1 Interval Coding

Interval coding is a very common and simple method of posture recognition, sometimes extended with error reducing algorithms. VPL used hardware and software look-up tables for min/max values which bracket the range of finger sensor values defining a posture, an interval coding. Each timestep the sensor readings from the glove were compared with the table, and if each of them fell into intervals corresponding to a certain posture it was recognized [ZLB$^+$87].

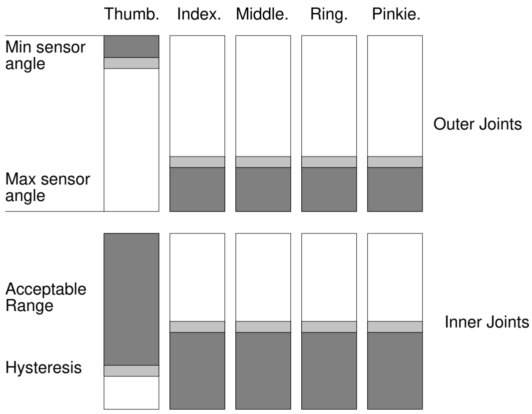


Figure 1.4: Interval coding for a hitch-hiking posture where the thumb is extended and the other fingers bent. The dark regions correspond to the acceptable range for sensor readings; if all sensor readings are in the dark regions a recognition is triggered. The light grey hysteresis regions are added to the acceptable range after a recognition, preventing "flip-flopping" between recognition and non-recognition due to noise in sensor readings close to the boundary of the acceptable range. (After [Stu92, Figure 19].)

For example, a pointing gesture would be ring-finger and thumb extended, and the rest bent. Hysteresis makes it possible to avoid sending repeated matches if the posture changes slightly by extending the intervals once a posture has been accepted, so that random movements will not bring the posture out of the recognition region.

This method is simple, understandable and flexible, but has some limitations. The range of each interval has to be rather large, up to 30% of the total range [Stu92] due to noise and the lack of precision in human gestures. The coding essentially divides the 10-dimensional space of postures into rectangular slabs, which often overlap if there are over 10 different table entries. Sturman claims this would be less of a problem with a more accurate device.

My personal impression is that it is very hard to do many potential postures since they require unusual hand configurations (cf. Figure 1.2) that do not mirror how we are normally trained to move our fingers, which means only a small subset of the space can be used. Affordance requires large slabs, which means that it is unlikely that increased precision would help.

Another, closely related method, is to code each recognized posture as average sensor values with standard deviations, and match if the measured sensor values fall within the appropriate standard deviations [Stu92]. One possible variant is to calculate a mean square error, and trigger a match when it is below a certain threshold. This would correspond to dividing posture space by ellipsoids, which may be less troublesome than slabs (in a high-dimensional space the corners of a cube will lie significantly farther from its center than the radius of the inscribed sphere, which increases the risk for intersection with other cubes).

A third interval method relies on the fact that most useful postures involve a combination of fully extended or flexed joints, and instead of a detailed look-up table only stores whether each joint should be extended, flexed or is irrelevant. If a sensor value is above 0.8 (normalized) it is regarded as flexed, and if it is below 0.2 it is regarded as extended. This method is robust and fast to process, and is ideal for hard-coded hand postures. Unfortunately most of the possible postures are hard to do or lack everyday meanings.

Interval methods are easy to code and often sufficient for simple applications or prototyping. They do not require much training to learn a gesture (it may already be hard-coded), and can easily be extended. The main drawbacks are that the number of recognized postures is low (< 10), the users often have to learn obtuse and unnatural gestures and many users feel frustrated by the lack of flexibility in doing the gestures. More complex methods can recognize more postures and more complex postures, but they require longer training times and are less flexible.

1.9.2 Statistic Similarity/Instance-based Learning

A method closely related to the sphere-division method mentioned above is discussed by Newby [New94]: a posture to be recognized is compared to a number of known prototype postures, and the Euclidean squared distance is calculated. The prototype with the least distance from the posture is selected as the match (if the distance is less than some error distance). It is pointed out in [New94] that this method allows a much greater number of recognized postures than interval coding; this is due to a more efficient division of the posture space into a Voronoi partition[7], where two regions by definition cannot overlap. As we will see in section 2.3 this is essentially a form of Vector Quantization.

Using this method Newby achieved reliable recognition of practically all of the 36 postures (ASL letters and numbers) in his system with confusion only between postures that differ only in movement.

[7] A Voronoi partition of $\mathbb{R}^n$ with respect to a set of k points x_i is a partition into k regions where the regions are defined as all points in the space closer to the point x_i than any other point x_j. The resulting regions will be n-dimensional (possibly infinite) convex polyhedrons.

1.9.3 Static Neural Networks

Another obvious method is to collect an example database of gestures and use it to train an artificial neural network. This avoids having to define each gesture in detail. It is hoped that the network will find the important features and abstract them. In addition, the networks are often robust to noise and manage to divide posture space into (Bayesian) optimal volumes.

The main drawback is of course the extensive training that must be repeated when introducing new gestures, and possibly for each user. It might be possible to employ continuous learning during use of the recognizer, which would make it adapt to the users and their tasks. Other problems are the risk for overlearning, where the network becomes overly specialized on the training data, loosing its ability to generalize (this can be avoided by careful testing), that they tend to consume much processing power and the problem of finding suitable training data; often the training data must be labeled for supervised training, but finding a good labeling of gesture data is equivalent to solving the recognition problem! Often labeling is done by hand (in the case of Fel's Glove-Talk system 30356 patterns were hand-labeled).

Kalawsky [Kal93] suggests a simple backpropagation network for static posture recognition using the VPL DataGlove using 10 input units, 8 hidden units and 8 output units representing the different posture-classes. This is quite similar to the posture recognition module used in the GIVEN system [VB93].

Beale and Edwards [BE92] used a multilayer perceptron to classify input into five postures (the wovels *a, e, i, o* and *u*). While the starting network had eight units in the hidden layer, it was pared down to three. The most interesting result was the suggestion that the energy landscape for the data set was relatively simple and unconvoluted [Wat93]; this suggests that at least for non-overlapping postures neural networks may be relatively simple to train.

In Fels' Glove-Talk system [Fel94], several backpropagation networks and radial basis functions were used to control speech synthesis. He achieved 92% success rate at recognizing 203 signs based on 66 hand shapes combined with 6 gestures.

Also worth mentioning is the work by D.S. Banarse on posture recognition using a neocognitron network[8] and an image grabbing program [Ban93]; although its recognition ability was rather limited (open, closed and pointing postures) it shows that visual recognition of hand postures through neural networks may be feasible.

1.9.4 Linguistic Methods

Several linguistic or syntactical methods of gesture recognition have been proposed and used, most based on formal grammar trees.

The GLAD-IN-ART project has used a feature tree model, where key features are represented by sub-branches and as a gesture is performed tree-traversal takes place depending on the identified features and when a leaf node is reached the gesture will be fully classified [Wat93]. A gesture can be represented in

[8]The neocognitron is a biologically inspired visual recognition network, whose architecture is modelled after the retina and visual cortex [FS91, chapter 10].

several leaf nodes, allowing alternative ways of making it. Postures can also be defined using trees, allowing fast classification.

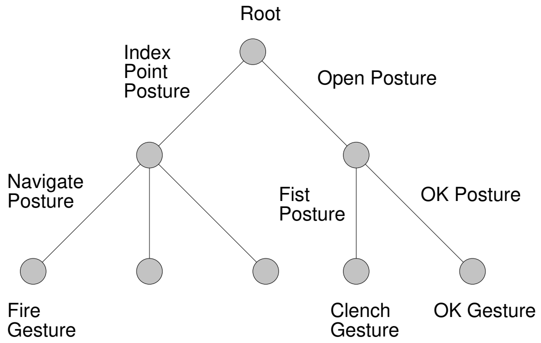


Figure 1.5: Gesture recognition using tree-traversal. Postures correspond to branches and gestures to the leaf nodes (after [Wat93]).

Hand, Sexton and Mullan [HSM94] used a formal grammar model for gesture recognition. They used hand postures and movement components as basic elements, and formed a formal grammar of elements and gestures. Their system recognized 15–75% of all gestures, with large variations between different gestures, although the mis-classification percentage was in general quite low. Mainly their results seem to have been due to the poor resolution of their glove (a Mattel PowerGlove) and somewhat narrow limits on allowed movements.

1.9.5 Starting and Ending Postures

A compromise between static and dynamical recognition is to recognize a gesture by its starting and ending postures, such as in the virtual environment interaction system described by Jacoby et al. [JFH94]. This requires some analysis of the gesture which seems hard to automatize, due to the segmentation problem.

1.9.6 Configuration States

An actual gesture can be seen as a trajectory through a configuration space, the space of all possible hand positions and postures (in the case of 16 input channels it would be a subset of $\mathbb{R}^{16}$). The configuration space can be divided into fuzzy configuration states, corresponding to more or less distinguishable postures. A gesture (in the abstract sense) can be seen as a ordered sequence of configuration states, and a trajectory can be classified by which states it passes through [WB95]. This removes the explicit time dependency, since only the curve in configuration space is relevant. The transitions between different states can also be seen as a natural segmentation of the gesture.

1.9.7 Feature Analysis

Feature analysis divides the gesture into a number of features: starting and ending postures, mean direction of movement, movement distance, bounding volume of the gesture in space, wrist rotation, Fourier components, number of zero-crossings of the derivatives or histograms of the various parameters. These features are then classified, either using instance-based learning (see section 1.9.2), weighing together their strengths [Rub91] or by using decision trees [Qui92] (cf. section 1.9.4).

Using feature analysis of 95 sign language signs made using a PowerGlove by different users Kadous achieved 80% recognition, which is surprisingly high given the lack of precision of the hardware [Kad95]. Baudel and Beaudouin-Lafon used a similar method to control a presentation with a DataGlove, achieving 72–98% recognition of the 50 gestures used (depending on training and hand size) [BBL93].

1.9.8 HMM

HMM, Hidden Markov Models are very popular in speech recognition and have also been used for gesture recognition [LX96, SP90].

An HMM models the system as a Markov process moving between different (hidden) states, emitting observable symbols on transition from one state to another. It can be represented by the three matrices $\lambda = (A, B, \pi)$. The matrix $A = a_{ij}$ specifies the probability that the internal state will change from i to j. $B = b_{jk}$ represents the probability that the system will generate the observable output symbol k on transition to state j. π is the probability distribution of initial states for the hidden Markov process.

When used for gesture recognition, the problems are: (1) to determine the probability that a given sequence of observable symbols would be generated by a given HMM λ, and (2) to generate an HMM that explains a set of sequences of observables. To recognize a gesture the system compares the likelihood that the component observables (often a series of vector quantized states) were generated by a list of different models, and returns the most likely classification. One of the main reasons HMMs are used is that they act as dynamic time warping, removing the details of the time evolution of the gesture while keeping the information about what postures have been formed.

Yanghee and Wohn [YW96] used an HMM to recognize and segment the 3D hand movement pattern of a gesture (projected onto a plane fitted to the movement to reduce dimensional complexity), achieving around 0.9% errors on test data consisting of 10 gestures.

Lee and Xu [LX96] used hidden Markov models to recognize 14 different sign language alphabet gestures. By using the iterative Baum-Welch algorithm to train the HMMs the system learns incrementally during use and new gestures can be added interactively. Their system achieves 1% error after just 2 examples and 0.1% error after 4 examples. The main limitation on their system was that the number of gestures that can be classified were limited by the number of observable symbols the preprocessor (a FFT combined with vector quantization) could generate.

1.9.9 Dynamic Neural Networks

Traditionally neural networks have not represented time well since all connections and neurons are assumed to be instantaneous or with a fixed delay; there is no natural, obvious way of representing temporal patterns in this kind of network.

Two groups of neural networks which have mainly been used to study spatiotemporal recognition are recurrent networks and TDNN, Time Delay Neural Networks.

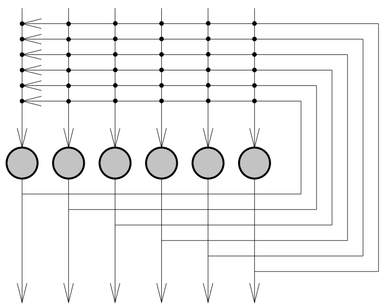


Figure 1.6: One layer recurrent network with full connectivity. Each unit is connected by a feedback connection to every other unit, and receives data from the outside.

In recurrent networks there exist feedback connections, so that output or hidden units also project as input units to the network (after a certain time delay) enabling memory for context and temporal patterns of activity (Figure 1.6). Adjusting the weights during training is nontrivial if the problem is not strictly defined, and the network can become unstable or exhibit spurious attractors.

In TDNN, the present data and data from times $t - \tau_0, t - \tau_1, \dots, t - \tau_k$ are presented to the network (where the delayed data may possibly have been filtered or modified; the input layer can be seen as k banks of shift registers), creating $k \cdot n$ input units, where n is the number of features in the data. This turns the dynamic input into static input which can be handled by an ordinary network. It is possible to combine several layers of delay banks into multilayer networks with hidden units, such as in Figure 1.7.

This architecture is simple to implement and training can be done using standard methods, but has some drawbacks. The most serious drawback is that it is not very good at recognizing a pattern that appears at a different speed than its training patterns (like a gesture made fast or slowly) if there are too many banks. The number of units also tends to be very large, and selecting a good time window is hard.

One interesting version of the TDNN is the Time Delay Radial Basis Function Network (TDRBFN), where the input banks project onto a layer of radial basis functions (see section 2.5) which then project to an output layer [Ber94]. This combines the temporal classification abilities of the TDNN with the fast training of RBF networks. This model is closely related to the hybrid network I have studied (cf. Section 2.6).

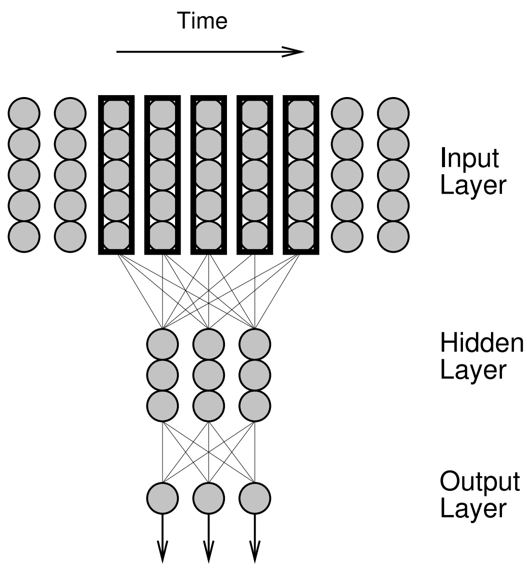


Figure 1.7: TDNN network with one hidden layer. Data is entered into the banks in the input layer, which are processed by the three banks of the hidden layer.

It should be noted that it is quite possible to combine recurrent networks and time delay networks; a popular class are the NARX (Nonlinear AutoRegressive models with eXogenous inputs) networks which are used in time series prediction [LGHK96]. NARX networks both receive delayed versions of their input signals (as in TDNN) and their own output signals.

Both TDNN and recurrent architectures have been used for speech recognition, and the TDNN was in fact introduced for this purpose [WHH$^+$89]. In addition there are neural networks where the connections act as IIR filters (this requires modified backpropagation training back through time), which gives the connections "memory" as previous signals will affect the strength of new signals, spatio-temporal self-organizing maps where activity moves from winning node to winning node [EP96] and various more complex models where the neurons have temporal properties such as spiking or act as leaky integrators.

Murakami and Taguchi [MT91] used a 403-node recurrent neural net to recognize 42 Japanese sign language postures. They achieved 98% correct recognition of gestures made by the people who had trained the network and 77% recognition of users not in the training set. They also demonstrated an independent 10 word gesture lexicon with 96% accuracy.

Martin Brooks used Kohonen networks [Koh84] for dynamic gesture recognition [Bro89]. He used several fairly small networks (around 20 neurons), one for each gesture, that recognized paths through the posture space. During operation the networks concurrently read the DataGlove input and responded if they recognized the gesture. Success appears to have been moderate.

In the GIVEN system several different approaches to gesture recognition with neural networks have been tested. Väänänen and Böhm [VB93] used a TDNN-like network where data from the DataGlove is passed to two 5 bank TDNN after some preprocessing. One recognizes finger gestures, the other hand gestures (having 50 and 30 input neurons respectively). The hand gesture network does not look at the actual position and direction of the hand, but instead at their derivatives. This use of phase space information is an efficient way to get dynamic information without increasing computational complexity too much.

Böhm [BBS94] also used two backpropagation networks to filter input data "horizontally" and "vertically" (i.e. in the time and amplitude domain) before being analyzed by a Kohonen map. The Kohonen map is presented with the data in the TDNN fashion, as a matrix of banks of sensor values. To solve the problem of time variability several versions of the data are presented with different time compressions. In a more complex model described, the Kohonen map was used to identify gesture parts which were then recognized using a recurrent neural network. The approach seems to have worked well but required much computing power.

Chapter 2

Model, Network, Hardware and Software

2.1 Model

The model of gesture recognition I experimented with is a hybrid network: after initial preprocessing (Section 2.2) the hand data is passed to a radial basis function network (Section 2.5) trained using the CSL algorithm (Section 2.3.2), linked to a Bayesian neural network (Section 2.4) which does the classification.

The preprocessing seeks to make the data as uniform as possible; ideally noise, user dependencies and speed differences should be removed or limited and the resulting data filtered in suitable ways.

The radial basis function network seeks to recognize postures or features of gestures. It is trained in an unsupervised manner on the preprocessed data. In order to make the network able to deal with temporal information several different approaches are studied, including adding derivatives and TDNN-like input.

The resulting feature activation values are in turn fed into a Bayesian net, which calculates the estimated probabilities of the gesture classes. The class with the highest probability estimate is chosen as the likely gesture class (in the case that there are no likely gestures the network defaults to the "no gesture" class).

2.2 Preprocessing

Let $x^{\mathrm{raw}}(t)$ be raw data from the glove, consisting of a vector of channel data $x_i^{\mathrm{raw}}(t)$.

The first step in preprocessing is to normalize the glove values, so their mean becomes zero and their variance 1:

$$x_i^{\mathrm{norm}}(t) = a_i x_i^{\mathrm{raw}}(t) + b_i$$

where a_i and b_i are suitable rescaling coefficients calculated beforehand.

The second step is filtering the glove data to remove noise and make the values more stable. This is done using an IIR (Infinite Impulse Response) filter of the form

$$x_i^{\mathrm{IIR}}(t) = kx_i^{\mathrm{norm}}(t) + (1-k)x_i^{\mathrm{IIR}}(t-1)$$

where k is a filtering constant.

The IIR filter is a lowpass filter, and this will make the glove feel sluggish if the time-constant is too large, so finding a suitable value is important.

A third optional step consists of moving from time parameterized data to arc-length parameterized data (Figure 2.1). This removes differences in execution speed.

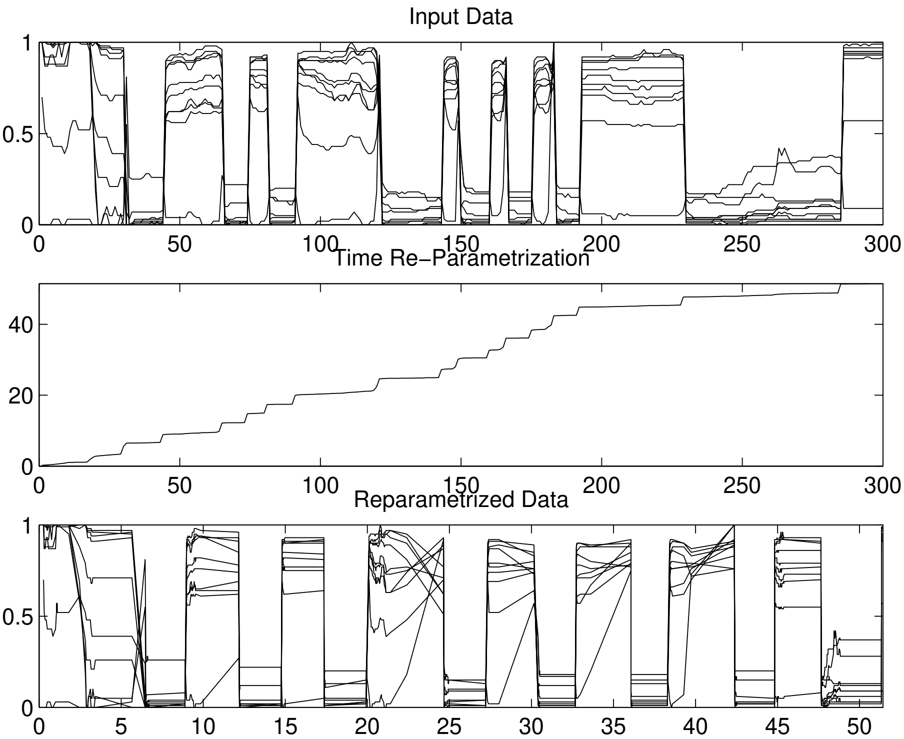


Figure 2.1: Time reparameterization of glove data. The topmost panel contains typical glove data, consisting of several similar gestures of unequal length and speed of execution. A reparameterization of the time coordinate $t \to t'$ based on an approximate arc-length in posture space (equation 2.2) is shown in the middle panel; horizontal stretches correspond to periods during which the hand doesn't move, while the "steps" correspond to fast movements. The lowest panel shows the same data after reparameterization. Note that the gestures are now of roughly equal length (eg. the three short gestures in the interval $t = 150$–200 are now mapped to $t' = 25$–40 and are just as long as the others). The last plateau has been compressed into a very small space near $t' = 50$.

My method works as follows: the distance $S(t)$ in posture space between the current and previous position is calculated as

$$\Delta x_i^{\mathrm{IIR}}(t) = x_i^{\mathrm{IIR}}(t) - x_i^{\mathrm{IIR}}(t-1)$$

$$S(t) = d\sqrt{\sum_{i=1}^{n} c_i \left(\Delta x_i^{\mathrm{IIR}}(t)\right)^2}$$

where c_i are weighting constants, equal within each category of data (finger data, position data or direction data) and d is a scaling parameter. The values of c_i have to be adjusted so that both finger flexion and hand movements are given suitable weights, given the fact that they are measured differently[1]. d is adjusted so that the average growth of $S(t)$ is roughly 1.

$S(t)$ is then used as a re-parameterization of the data, essentially an approximate arc-length parameterization along the measured trajectory in posture space. If the values $x^{\mathrm{IIR}}(t-k)$ are desired, they can be calculated using the following algorithm (where each measured value $x^{\mathrm{IIR}}(t)$ is assumed to occur at integer times i).

The Timewarp Algorithm

1. Set $T = 0$ and $i = 0$.
2. If $T < k$, $T := T + S(t-i)$, $i := i+1$ and go to 2.
3. Otherwise return

$$x^{\mathrm{warp}}(t-k) = (1 - T + k)x^{\mathrm{IIR}}(i) + (T-k)x^{\mathrm{IIR}}(i-1)$$

The optimal length of the time window is one of the parameters that has to be determined experimentally, see the results in section 3.4.

The result of all this processing is data that are uniform in time and amplitude, with noise averaged away. The gesture, which can be seen as a curve in posture space, is now arc-length parameterized.

2.3 Vector Quantization

Vector Quantization (VQ) is a transformation T from a k-dimensional Euclidean space $\mathbb{R}^k$ to a finite subset X in $\mathbb{R}^k$ made up of N vectors, usually called the codebook or the set of representative vectors; each input vector x is replaced by a codebook vector $T(x)$. The transformation can be seen as lossy data compression or a classification task, and has been used in image compression, speech recognition and as the basis for several neural network models. One of the most

[1]It should be noted that the position and direction data from the Polhemus sensor has to be manipulated or ignored to avoid sudden discontinuities due to zero-crossings.

useful properties of the encoding is that the codebook vectors are mutually exclusive, which is especially useful for input into a Bayesian neural network.

A common optimization is the use of product codes, where $\mathbb{R}^k$ is split into a Cartesian product $\mathbb{R}^{a_1} \times \mathbb{R}^{a_2} \times \cdots \times \mathbb{R}^{a_l}, \sum a_i = k$ and VQ applied independently in each factor space with different a_i dimensional codebooks. The code is the concatenation of the resulting vectors. This makes it possible to separate independent factors at an early stage, and makes finding the codebooks more efficient.

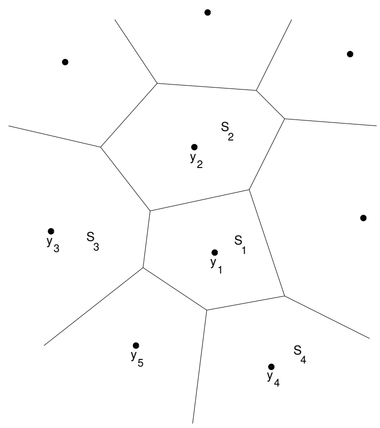


Figure 2.2: The partitioning of $\mathbb{R}^k$ into regions S_i by the vector quantization operator T. All vectors inside a region will be mapped to the closest codebook vector y_i.

$\mathbb{R}^k$ is partitioned into N regions by the codebook. If $y_i = T(x)$ is a codebook vector then its partition is the region $S_i = T^{-1}(y_i)$, the region where all vectors are mapped to y_i.

The choice of the codebook vectors and the transformation is the central problem; an obvious constraint is that one wants to minimize the total distortion of all input vectors x (which usually have a complicated distribution in $\mathbb{R}^k$) as they are transformed into $T(x)$, usually measured as the Euclidean distance $d(x, T(x)) = |x - T(x)|$, given N. This is a hard optimization problem, since T is discontinuous and only takes a finite number of values, and in real applications the distribution $\rho(x)$ of input vectors is not known except for a set of samples $x_1, x_2, \ldots, x_M$.

Two necessary conditions for an optimal solution are known:

1. The partition of $\mathbb{R}^k$ by the codebook must be a Voronoi partition, i.e. each vector is represented by the closest vector in the codebook, or
$$T(x) = y_i \Rightarrow \forall j : d(x, y_i) \leq d(x, y_j)$$
2. For a given partitioning and Euclidean distortion each codebook vector y_i has to be equal to the centroid of the input vectors lying in S_i:
$$y_i = \frac{1}{|S_i \cap \{x_j\}|} \sum_{x_j \in S_i} x_j$$

2.3.1 LBG

There are several ways of finding an acceptable codebook. The classic method is the LBG method (named after Linde, Buzo, Gray [LBG80]).

The LBG algorithm

1. Starting with an initial codebook Y and a training set X:

2. Divide the training set into N subsets according to which codebook vector they lie closest to:

$$s_i = \{x_j \in X : d(x_j, y_i) \leq d(x, y_k), \forall k\}$$

3. Update the codebook vectors to be the centroid vectors of their respective sets:

$$y_i = \frac{1}{|s_i|} \sum_{x_j \in s_i} x_j$$

4. Compute the average distortion,

$$D(m) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y^{(m)}} d(x, y)$$

   and stop if the relative reduction in distortion is beneath a threshold value:

$$\frac{D(m-1) - D(m)}{D(m)} < \epsilon$$

   otherwise go to step 2.

Once the codebook vectors have been determined, $T(x)$ is defined as the codebook vector with the least distance to x.

The LBG algorithm will always reach a local minimum for the distortion function, but it is sensitive to the choice of initial codebook and often gets stuck in high-distortion minima. Often this is due to a high density of codebook vectors in a few clusters of training vectors, and very few (if any) in other clusters, vectors that are never used or vectors stuck between two clusters "belonging to it" since their mutual centroid lies there.

Since a good initial codebook can greatly enhance the performance of this algorithm, several methods to create it have been developed. The random code generation method selects N random vectors from the training data as the initial codebook.

Product codes can be used to calculate an initial codebook [LBG80]. The data is first quantized as scalars into a scalar codebook C_1. $C_1 \times C_1$ is then used as the initial codebook for a two-dimensional quantization resulting in a two-dimensional codebook C_2. $C_2 \times C_1$ is then used for a three-dimensional quantization, resulting in a codebook C_3, and so on until C_k is reached.

In the *splitting method* code vectors are successively split, subdividing their domains. The initial codebook just consists of one vector, which is moved to the centroid of the whole distribution of x. It is then split into two new code vectors $y_0 \pm \epsilon$ (where ϵ is a small random vector), which are moved to the centroids of their respective domains. These are in turn divided, until a sufficient number of codebook vectors have been calculated.

Another VQ algorithm, the *Pairwise Nearest Neighbor* (PNN) algorithm, works in a bottom-up fashion: at each step the two training vectors that are closest together are replaced by a single vector located at their mean. This process continues until N vectors remain; these are chosen as the codebook. This is a useful algorithm for determining a good initial codebook for LBG [Gon94, Appendix E, section 2a].

Related to VQ is the *competitive learning* (CL) algorithms that have been developed in neural networks. In these learning rules the neurons (usually called units) correspond to codebook vectors. Training consists of presenting a randomly chosen training vector and calculating the distance to it from the units; the unit with the least distance is the 'winner'. The winner is moved slightly towards the training vector (the 'winner-takes-all-rule'). As this is iterated, the units move to positions corresponding to the two necessary conditions discussed in section 2.3 for a local minimum. It has been shown [UN94] that CL is equivalent to LBG and has comparable or better performance since it sometimes can escape unstable local equilibria due to its stochastic nature.

2.3.2 CSL

It has been shown [UN94] that in general each partitioned region should have the same local distortion to minimize the expected distortion. CSL, Competitive Selective Learning, combines CL and an equidistortion principle in order both to minimize the total distortion and distribute it evenly over the codebook vectors. Vectors in regions of low or high distortion are occasionally selected and moved to regions with less extreme distortion. Since the selection is non-local it makes the algorithm less sensitive for the initial codebook, and the distortion is divided equally between all units.

The CSL algorithm (after [UN94])[2]:

The CSL Algorithm

1. Initialization: Given N, the number of units, a training set $X = \{x_1, x_2, x_3, \ldots, x_M\}$, a convergence threshold ϵ, and a set of initial weight vectors of units $Y^{(0)} = \{y_1, y_2, \ldots, y_N\}$. Set $m = 1$ and $D(0) = \infty$.

2. Training: For all $x \in X$, perform (a) and (b) like the conventional competitive learning algorithm.

   (a) Find the closest unit y_c to x

   $$d(x, y_c) \leq d(x, y_i), \forall j \neq c$$

   (b) Adjust the weight vector of y_c:

   $$y_c(t+1) = y_c(t) + \eta(t)[x(t) - y_c(t)]$$

3. Selection step: If $s(m) > 2$, choose $s(m)$ units to be selected and perform the selection algorithm (see below) for the units, increase m by one and go to step 2. Otherwise go to step 4.

4. Convergence test: calculate the average distortion

   $$D(m) = \frac{1}{|X|} \sum_{x \in X} \min_{y \in Y^{(m)}} d(x, y)$$

   If

   $$\frac{D(m-1) - D(m)}{D(m)} < \epsilon$$

   halt with $Y^{(m)}$ as the final weight vectors, otherwise increase m by one and go to step 2.

The number $s(m)$ in step 3 is the number of units that are subject to the selection process, and decreases monotonically as m increases. The $s(m)$ units are chosen alternately from the first and last rank of the subdistortion, to reduce the variance of the subdistortions.

[2]While conceptually simple, the actual algorithm is somewhat messy.

The selection algorithm is as follows:

The CSL Selection Algorithm

1. Compute a normalized fitness measure using the subdistortions D_j:

$$g_j = \frac{D_j}{\sum_{j=1}^{s(m)} D_j^{\gamma}}$$

where $\gamma < 1$ is a nonnegative constant.

2. Determine the number of units to be reproduced:

   (a) Compute $u_j = \lfloor g_j s(m) \rfloor$

   (b) For top $s(m) - \sum_{j=1}^{s(m)} u_j$ units with respect to the value of $g_j s(m) - u_j$, add one to each u_j.

3. For every j, reproduce u_j neurons by adding random perturbation vectors $\delta_{jl} (l = 1, 2, \ldots, u_j)$ to y_j. Halt.

The value of γ influences how much selection should take place, a high value will lead to excessive selection when the variance is large but the selection isn't very sensitive to changes in γ; a reasonable value appears to be $\gamma = 0.5$.

In step 2 u_j denotes the number of units that should be reproduced from the jth unit by the selection process. If $u_j = 0$ then unit j should disappear.

2.4 Bayesian Neural Networks

In a Bayesian neural network, the units represent stochastic events and their states are related to the probability of that event given the events corresponding to the already activated units. Input data correspond to known events, and the final state after a number of iterations corresponds to a decision or classification given these conditions. The original network of [LE89] was a one-layer Hopfield type network with autoassociation; a more complex multilayer structure that can deal with correlations, uncertainties and gradings in the input data is described in [HL93]. Relaxation allows the network to reduce noise, complete patterns and do hierarchical clustering.

Training is based on the Bayesian learning rule, which is a Hebbian rule: correlated units will be linked by positive weights, anticorrelated units will be linked by negative weights and uncorrelated units will be linked by zero weights.

The network can be used as a classifier network. Given some features, it deduces the probabilities for the different classes and usually the highest probability class is then used as the answer. This is equivalent to estimating the probability distribution over a random variable Q representing the probabilities of different classes q, given another variable X representing the measurement of some features. Bayes rule gives:

$$P(Q|X) = P(Q)\frac{P(X|Q)}{P(X)}$$

If X consists of the set $\{X_1, X_2, \ldots, X_n\}$ of independent variables the rule can be written as:

$$P(Q|X) = P(Q)\frac{P(X|Q)}{P(X)} = P(Q)\prod_i \frac{P(X_i|Q)}{P(X_i)}$$

Taking the logarithm we can rewrite the rule as a sum:

$$\log(P(Q|X)) = \log(P(Q)) + \sum_i \log\frac{P(Q, X_i)}{P(Q)P(X_i)}$$

By using a complementary coding of binary events this can be turned into a neural network model (Figure 2.3). One unit in the complementary pair represents the presence of an event, one represents its absence (activation zero of a Bayesian unit corresponds to a state where we do not know anything about an event, rather than knowing it has not occured). The weights from the positive unit correspond to the change in the network if there is positive evidence for the event while the weights from the negative unit correspond to the change if there is negative evidence.

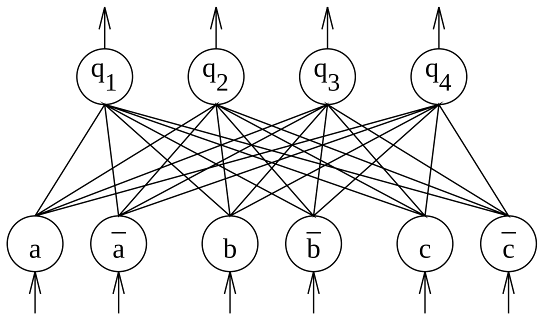


Figure 2.3: Two-layer Bayesian net determining the probabilities of four classes q_i from three features a, b and c, represented using complementary coding. (Figure is adapted from an original by Anders Holst.)

Given the set of features $A = \{x_i, x_j, x_k \ldots\}$ we can calculate the probability of a specific class q given these features as:

$$\log(P(q|A)) = \log(P(q)) + \sum_i \log\left(\frac{P(Q, X_i)}{P(Q)P(X_i)}\right) o_i$$

where o_i is the activity of an input unit, 1 if $x_i \in A$, and 0 otherwise. This equation is analog to the standard equation for a neural network unit [LH96],

$$s_q = \beta_q + \sum_i w_{qi} o_i$$
$$o_q = \pi_q(s_q)$$

where s_q is the support value of unit q (corresponding to activation in an artificial neural network context), β_q is its bias

$$\beta_q = \log(P(q))$$

(representing the a priori probability of the class), w_{qi} the weights

$$w_{qi} = \log\left(\frac{P(q, x_i)}{P(q)P(x_i)}\right)$$

(representing the correlation between the class and the feature) and an exponential transfer function clipped to avoid probability estimations larger than 1:

$$\pi_q(s_q) = \begin{cases} 1 & s_q \geq 0 \\ \exp(s_q) & \text{otherwise.} \end{cases}$$

Note that there is no difference between classes and features in this model, they are all treated as stochastic events and the weight matrix is symmetric $w_{ij} = w_{ji}$.

Training the network is done in one pass over the training set. The probabilities $P(x_i)$ and $P(x_i, x_j)$ are estimated from the training examples, and then used to set the weights and biases. This is done via counters in the units and weights:

$$C = \sum_\alpha \kappa^{(\alpha)}$$

$$c_i = \sum_\alpha \kappa^{(\alpha)} \pi_i^{(\alpha)}$$

$$c_i = \sum_\alpha \kappa^{(\alpha)} \pi_i^{(\alpha)} \pi_j^{(\alpha)}$$

where α is the pattern number, $\kappa^{(\alpha)}$ is the strength of pattern α and $\pi_i^{(\alpha)} = P(x_i^{(\alpha)})$ is the proportion of the population described by pattern α having attribute x_i. These counters are calculated in one pass over the training set, and then the probabilities of an event are estimated:

$$\hat{p}_i = \frac{c_i}{C}$$

$$\hat{p}_{ij} = \frac{c_{ij}}{C}$$

Care must be taken to deal with zero probabilities, since a logarithm of these values will be used. In practice such values are replaced with a negative value smaller than any other value used in the network, which appears to work well [HL93].

Weights are set to

$$w_{ij} = \begin{cases} 0 & c_i = 0 \vee c_j = 0 \\ \log(1/C) & c_{ij} = 0 \\ \log\left(\frac{c_{ij}C}{c_i c_j}\right) & \text{otherwise.} \end{cases}$$

and biases to

$$\beta_i = \begin{cases} \log(1/C^2) & c_i = 0 \\ \log(c_i/C) & \text{otherwise.} \end{cases}$$

2.4.1 Higher Order Networks

The above model assumes that the features are independent, both unconditionally and conditionally given q:

$$P(X) = P(X_i) \cdot P(X_j) \cdot P(X_k) \cdots$$

$$P(X|Q) = P(X_i|Q) \cdot P(X_j|Q) \cdot P(X_k|Q) \cdots$$

In real applications this is not true in general, although the naive network works well in many real world situations, especially since knowing which class has the highest probability is enough for correct classification, so the exact class probabilities are not necessary.

One solution is to pre-process the input data to reduce the correlations, for example through VQ methods. Another way is to extend the model into a multilayer network where the interdependencies can be split up between units.

If two features X_i and X_j are not independent, then they can be merged into a joint variable X_{ij}. The probability of X becomes:

$$P(X) = P(X_i, X_j) \cdot P(X_k) \cdots$$

and similarly for $P(X|Q)$. This adds new units to the network, one for each combination of outcomes of the original variables X_i and X_j. If the features $A = \{a, \overline{a}\}$ and $B = \{b, \overline{b}\}$ are not independent, their units are replaced by a group of complex units $AB = \{ab, a\overline{b}, \overline{a}b, \overline{ab}\}$.This kind of group, containing all possible combinations of outcomes of primary features, is called a hypercolumn and is naturally decorrelated.

The modified form of the Bayesian neural network consists of an input layer of complementary units, a hidden layer of complex (hypercolumn) units and an output layer of class units (Figure 2.4). The hidden layer and the class layer are connected by a normal Bayesian network. The activity of the complex units can be calculated by multiplying the activities of the involved input units. If the entire network is run as feedback network, there are also Bayesian connections between the hidden layer and input layer, and the activities of the input units can be calculated by adding the activities of the relevant complex units (i.e. $P(a) = P(ab) + P(a\bar{b})$).

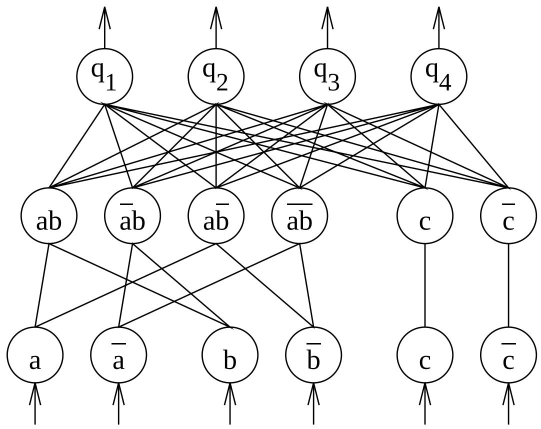


Figure 2.4: Higher order Bayesian network with one hypercolumn consisting of the four combinations of a and b and one simple column c. (Figure is adapted from an original by Anders Holst.)

2.5 Radial Basis Function Networks

A radial basis function is a function $f_c(x)$ which is only dependent on the distance of the point x from the point c:

$$f_c(x) = \phi(|x - c|)$$

where ϕ can be a step function, a Gaussian e^{-r^2/σ^2} or any other function suited to the model that decreases (or increases) monotonically with distance from c. Usually they have a characteristic distance called their radius.

Radial basis functions can be combined into networks, usually single-level networks where the units correspond to radial basis functions with different centers, radii and weights. Input data is presented to the RBF layer, where each unit calculates the functional response. These responses can then be weighted together by an output layer, making the RBF network able to act as a general approximator, classifier or predictor.

Training a RBF network consists of changing the centers c_i and radii r_i of the functions and their weights w_i to approximate the desired output. This can be done using unsupervised training with VQ methods. After the functions have been determined, the weights are set using supervised training.

The input to output map is found using local approximators, each unit activates only when there are stimuli in its region. The weights between the RBF layer

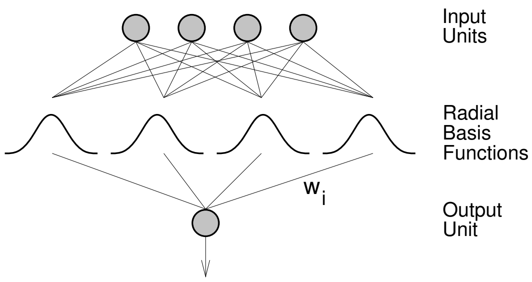


Figure 2.5: A RBF network.

and the output layer are few and can be trained quickly, which makes RBF networks practical for many applications. On the downside, the number of RBF units necessary to cover the input space may be large.

2.6 A Combined CSL and Bayesian network

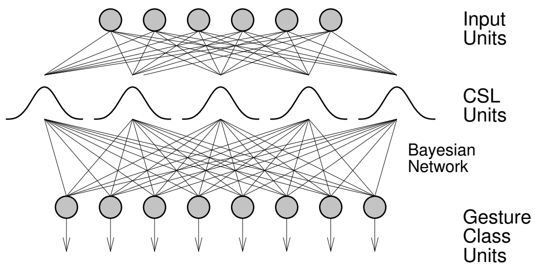


Figure 2.6: The hybrid CSL-Bayesian network. Input data is presented to the CSL layer which acts as receptive fields for the Bayesian classifier layer.

I have experimented with a hybrid approach, using CSL to locate the units of a radial basis function network used as input layer to a higher order Bayesian network. The CSL-RBF units act as "receptive fields": they respond in a Gaussian way to the distance to the input vector (where the width of the receptive field is equal to its mean distortion). This input is in turn processed by the Bayesian network, that maps the RBF activations into gesture probabilities. Ideally, the RBF level will find relevant features in the raw data, and the Bayesian level will find the links between the features and gestures.

The product space structure used in vector quantization works well together with the hypercolumn structure of the Bayesian net, since it allows reduction of correlations even before the Bayesian hypercolumns. The k-dimensional input data is divided into a number of lower-dimensional input spaces where RBF units are distributed using CSL. The signals from these units then form a hypercolumn in the Bayesian network. Not only does this decorrelate the input data, but it also speeds up training significantly.

2.7 Hardware

Figure 2.7: The VPL DataGlove (center) at IPLab. The small black box to the left of the glove is the optoelectronics sending and measuring the light through the optic fibers in the glove. It is in turn connected to the control unit, the large black box behind the glove. The control unit is also connected to the Polhemus transmitter (the small grey device to the right of the glove).

In my work I used the VPL Dataglove at IPLab to gather gesture data. It is a medium-sized dataglove, with the 10 basic flexion sensors and a polhemus tracker, giving 16 input data channels updated at 15 Hz. Each data packet consists of 10 bytes of flexion data and six 2-byte floating point position and direction data. The polhemus sensor works inside a hemispherical volume of around one meter radius; when it moves through the plane it begins to report negative coordinates instead, causing a discontinuous transition.

One problem with the setup on IPLab was a hardware error that intermittently caused severe spiking. This can be filtered away by postprocessing, which ignores values which differ too much from previous values and replaces them with values extrapolated from previous values but in my work I simply avoided such noisy data.

For processing of the data I used a Silicon Graphics Indigo for data gathering, and various Sun SPARCstations for neural network training and testing.

2.8 Software

To get realistic glove data (and avoid having to write too much code), I used the device drivers and event handlers of the DIVE (Distributed Interactive Virtual

Environment) system, an Internet-based multi-user VR system developed at SICS (the Swedish Institue of Computer Science) [Hag96].

The device drivers and event handlers I mainly used were developed by Johannes Keukelaar as part of his master's thesis [Keu95]. Essentially, I just extracted the raw data from the glove device for external manipulation.

Although my original intention was to create a postprocessor device for the event handler system, problems with internal delays in the operating and DIVE systems, integration of neural network code running on other computer architectures and frequent version changes in the DIVE project would make this too cumbersome.

The neural network program used in this work, MCSL, was written by Anders Lansner. It implements the CSL training of a RBF layer linked to a Bayesian network. Minor adaptions were made, but pre-processing was implemented in a stand-alone data management and processing system (Wake) by me. I also wrote a program in Matlab (Ocean) to mark and concatenate the raw data.

2.9 Implementation

2.10 Gesture Classes

The gesture classes I used in my experiments were arbitrarily selected; I have not made any attempt to mimic the gesture commands in any real system since they tend to be affected by the system recognition ability and application rather than being general gestures. Several of these gestures were suggested by friends when I described the problem of interaction in virtual reality.

The gestures mix both postures, simple gestures (a posture with some movement, usually change of hand direction or finger flexion in a single direction) and complex gestures (changes in flexion and direction with several components). See Table 2.1.

Table 2.1: Gestures used in my experiments. P = Posture, S = Simple gesture, C = Complex gesture.

#	Name	Type	Description
1	No gesture	P	A flat hand or no special pose. This class is the "default class" that should be active when no other class is active.
2	Blessing	P	A blessing sign (index and middle finger extended, the other fingers bent into the palm). Might be used to acknowledge a system decision.
3	Come	C	A repeated beckoning gesture with the index finger, similar to the gesture mentioned in [JFH94]. Might be used to bring an object into the vicinity.
4	Heavy Metal	P	The Heavy Metal "goat" sign. Might be used to express emotion?

5	Explode	S	Hand closes into a fist which is quickly "exploded" into a spread hand. Might be used to remove or distribute an object.
6	Grab	S	Grabbing motions, the hand becomes a fist. Common method of designating an object for interaction or translation.
7	OK	P	The OK sign (thumb and index finger forms a circle, other fingers straight). Acknowledging a result or responding to a system yes/no question.
8	Open	C	The Open File gesture used in the movie *Johnny Mnemonic*: a grabbing motion, the fist is turned upside down and then slowly released as if letting something unfold in the palm. Might activate inherent behaviors in an object.
9	Pointing	P	Pointing gestures (the classic NASA flying gesture).
10	Cylinder	P	Thumb and index finger forms a circle, other fingers bent so that the hand forms a open cylinder. Might represent activation of a "telescope" tool.
11	Beg	P	A begging gesture (palm up, slightly curved). Might be used to demand an object from the system. This was chosen to see how much it would be confused with the plain "null gesture".
12	Thumbs	S	"Thumbs up" and "Thumbs down" gestures. Might be used to respond to queries from the system.
13	Walk	C	A "walking" gesture using the index and middle finger, hand pointing downwards. Might be used to move a vehicle or robot, similar to the application in [Stu92].
14	Y	P	The ASL sign for the letter 'Y' (thumb and little finger extended, others bent into a fist).

2.11 Data Gathering and Processing

The finger flexion data and hand position/orientation was recorded from the glove device driver and stored as datasets containing both gesture and non-gesture data. I then used a Matlab program (Ocean) to mark the gestures and concatenate the datasets into a standardized database.

Gesture data was sampled from myself and two other persons, and then integrated into the standard dataset used for the experiments. This dataset contains a total of 4671 input patterns, divided between the 14 gesture classes

(Figure 2.9). Class #1 was represented more than the others (28%) since it was much more variable than the other gestures and also important to identify well in order to limit false positives, cf. [BBS94].

When marking the gestures, I took care not only to mark the fully evolved posture or gesture (such as a fist) but also the move towards this state (the closing of the hand) since it should properly be considered a part of the gesture and was assumed to speed up recognition (since the system now could react as soon as a movement towards a known posture or gesture was taken), at the expense of having to deal with weakly formed gestures.

I also made a series of test gestures to study the dynamic responses of the network on a wholly unfamiliar data series; this dataset was used in Figures 3.3, 3.16 and 3.20.

For the various experiments I used another program (wake) to preprocess the data, select the gestures to be used in the test and training data, and create auxiliary information such as hypercolumn structures.

The training was done in all cases using a random division of the data into a training dataset and a testing dataset in an approximate 2:1 ratio. Validation was done by comparing the network's recognition ability of training patterns with the test patterns.

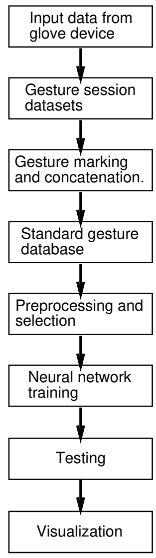


Figure 2.8: Experimental dataflow.

During training the training set of patterns were first used to train the CSL-RBF layer in two passes, and then the Bayesian network in one pass. The first CSL pass used selection operations with probability 0.1 and fast unit movement $\eta(t) = 0.01$ to place the units in roughly the right place, while the second CSL pass did not employ any selection operations and used $\eta(t) = 0.005$ to move the units gradually towards their optimal positions. Training of the Bayesian network was then done using the resulting CSL-RBF structure.

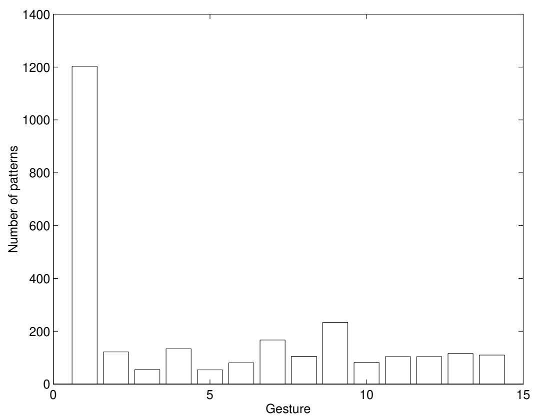


Figure 2.9: Number of patterns belonging to the different classes.

The output unit with the strongest activation was selected as the classification decision of the network and compared to the correct classification of the pattern. In these experiments there was no threshold for the output activations used in the classifications (if all activations are equally low it could be interpreted as a sign of non-recognition), since it was found that in general the activations were close to binary and with gesture class #1 as default active; only mis-classifications were relevant for analysing the network performance.

The resulting information about the average number of mis-classifications of the gesture classes and test data in general was used for various forms of visualization and analysis. When calculating average network efficiencies several trials were often used in order to calculate approximate error bounds. In a few cases network output activations were studied.

Chapter 3

Results

In the following, I will compute the results for different approaches to dynamic gesture recognition. The first is a static network, where the present glove data is presented to the RBF units and their output is fed to the Bayesian units. The second is a phase space network, where glove data and its derivatives is presented to the RBF layer (if the first network can be said to look at the configuration space of the glove, this network looks at the corresponding phase space). The third approach is a TDNN system, where the RBF layer receives information from the present and several snapshots of previous data.

3.1 Structure of the Glove Data

The finger data appear to be somewhat clustered in the 10-dimensional posture space (Figure 3.1), using most projections into 3D or 2D.

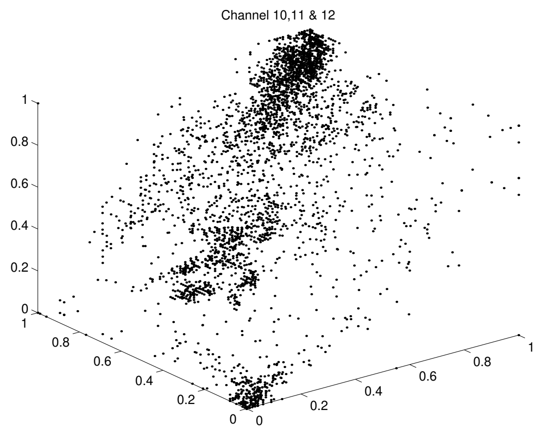


Figure 3.1: Distribution of postures based on channel 10,11 and 12 (MCP and PIP of index finger, MCP of middle finger).

The position and direction data mainly reflect the preferences of the users and their work, which suggests that it is not as relevant for recognition as the finger flexions; a posture should be interpreted in the same way over a fairly wide range of hand directions and positions (some postures change their meaning when redirected, such as the "thumbs up" gesture).

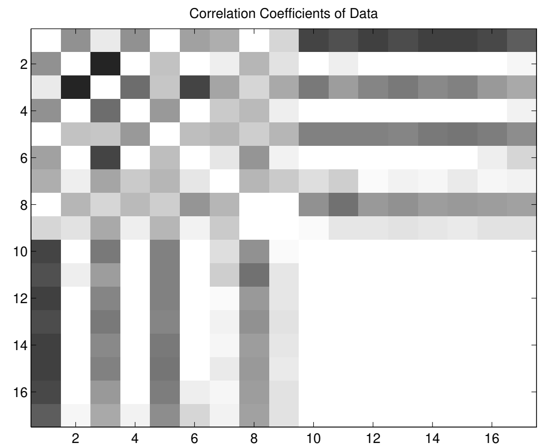


Figure 3.2: Correlations in the glove data. Dark shades correspond to negative correlations, white to positive correlations and medium grey to correlations close to zero. Channel 1 = gesture type, 2, 3, 4 = position, 5, 6, 7 = direction, 8–17 = finger flexion.

A statistical analysis of the gathered data shows that there are strong internal correlations (Figure 3.2). The finger flexions are strongly correlated with each other, while the correlations between the position and direction data is weaker. A mutual information estimate of the distribution along the lines in [HL96]

$$I(X,Y) = \sum_{x_i,y_i} P(x_i,y_i) \log\left(\frac{P(x_i,y_i)}{P(x_i)P(y_j)}\right)$$

(where an event in this case is defined as when a parameter is above its mean value) gives roughly the same result, which suggests a division of the input into three subspaces, one for position, one for direction and one for finger flexion.

3.2 Static Recognition

I trained the basic network to classify data from the VPL glove into the 14 classes of gestures and postures in Table 2.1. At each timestep one pattern was shown to the RBF layer and the Bayesian layer classified it depending on the activations of the RBF units. The response was nearly binary; most class unit activations were very close to one or zero.

As can be seen in Figure 3.3, the basic posture recognition method correctly identifies most classes most of the time, but has a tendency to give false positives to intermediate hand states (as when a gesture is being formed or after it has

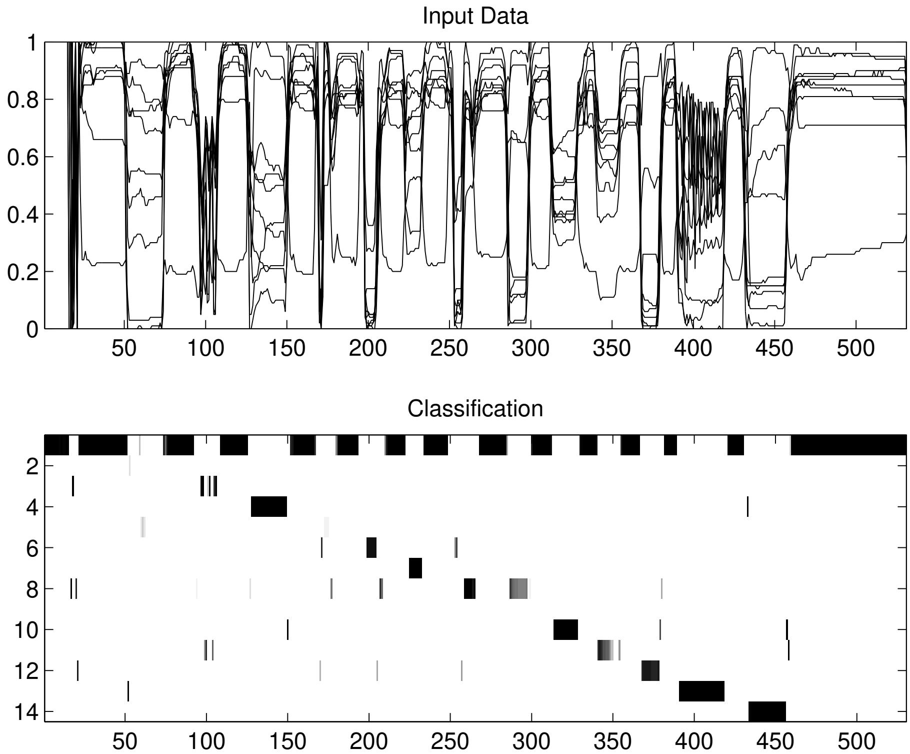


Figure 3.3: Network response to sample data. The upper plot shows the actual finger flexion input data as a function of time (1 corresponds to a non-flexed joint, 0 to a maximally flexed joint). Several gestures are obvious, such as #3 "Come Here" (around $t = 100$), #6 "Grab" (around $t = 200$) or #13 "Walk" (around $t = 400$).
The lower plot shows the activation of the output units corresponding to the classifications as a response to the data: dark shades correspond to strong activations, white to inactive units. As can be seen there exists spurious firings and some misclassifications, especially obvious for gesture #3 ($t = 100$) and #9 (at $t = 290$).

ended and is moving towards a non-gesture) or mis-classify postures due to noise or random motions. More importantly, it has trouble consistently recognizing gestures (see Figure 3.4), and often tends to shift its classification between the real class and one or more postures similar to part of the gesture[1]. This suggests that while the basic network is good at classifying postures, gestures require a network with temporal components.

When trained with only postures (8 classes), the mean performance and its standard deviation is $97 \pm 0.9\%$ for the training set and $96.8 \pm 1.6\%$ for the test set. When simple gestures are introduced (11 classes), the performance decreases to $95.6 \pm 1.05\%$ and $95.9 \pm 1.6\%$ respectively, and training using all gestures (14 classes) in the training set gives $93.5 \pm 1.05\%$ and $93 \pm 1.5\%$. To

[1]It should be noted that some of the problems with erratic recognition can be solved by integrating the response over time, for example by using an IIR filter.

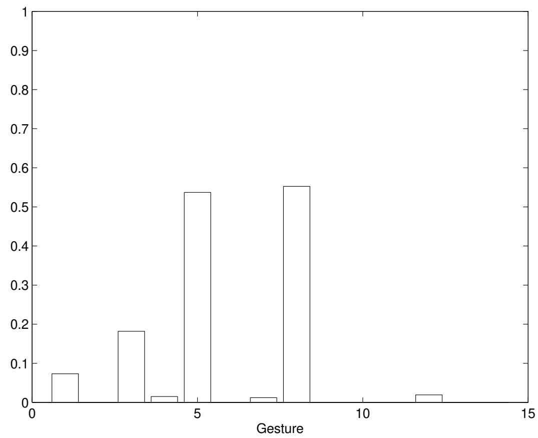


Figure 3.4: Misclassifications of different gestures when using the static network measured as the ratio of the number of erroneous classifications to the number of classifications for each gesture. Note how the network has trouble identifying gesture #5 and #8 ("Explode" and "Open"), both complex gestures. The correlation between sample size and the percentage of misclassifications is practically nonexistent (-0.0994). The mean percentage of misclassifications is $\approx 7\%$

check that this result was not due to a lesser number of classes to learn, I trained the network with a mixture of gestures and postures but with fewer classes, achieving a performance of $95.3 \pm 0.9\%$ / $94.3 \pm 2\%$ for 8 classes and $93.3 \pm 0.9\%$ / $93.1 \pm 1.5\%$ for 11 classes. With only complex gestures and the null gesture (4 classes) the performance was just $93.6 \pm 1.3\%$ / 93.5 ± 2.4.

Since the recognition percentage was consistently and statistically significantly ($p < 0.05$) lower for the mixed gesture sets than the posture only and posture + simple gesture sets, it is very likely that it is the gestures that decrease performance the most for this network, and especially the complex gestures.

3.2.1 Classification Structure

I have experimented with visualizing the decision boundaries of the static network on a part of a 2D plane in the 16-dimensional input space. I selected three arbitrary postures (#9, #10 and #11), calculated the average input vector $\bar{x}_i$ for each posture and interpolated simulated input data linearly between these three points in $\mathbb{R}^{16}$: $x(s,t) = \bar{x}_9 + (\bar{x}_{10} - \bar{x}_9)s + (\bar{x}_{11} - \bar{x}_9)t,\ s,t \in [0,1]$. The interpolated data was then fed to the trained network, whose output was visualized.

Figure 3.5 shows the classifications made by the network for the data in the region. As can be seen the decision boundaries are roughly a Voronoi partition, with some deviations along the border between #10 and #11 ("Cylinder" and "Beg"); the RBF-Bayesian structure does not force the boundaries to be straight, and they can presumably become very complex if there are enough RBF units in that region of feature-space.

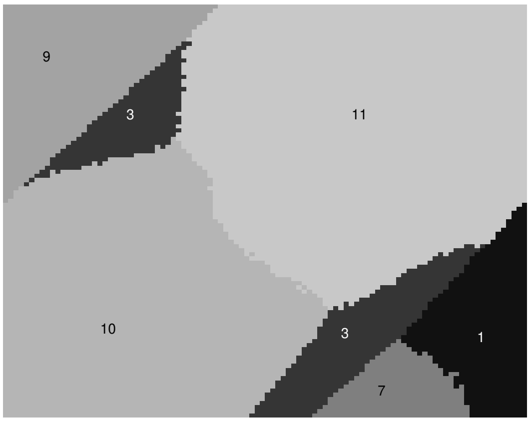


Figure 3.5: Classification structure on the hyperplane through the points $\bar{x}_9, \bar{x}_{10}$ and $\bar{x}_{11}$ (upper left corner, lower left corner and upper right corner). The selected classification was the output unit with the highest activation. #1 = "Null", #3 = "Come", #7 = "OK", #9 = "Point', #10 = "Cylinder", #11 = "Beg".

It is interesting to note how #3 "come" is represented in two places. These two domains correspond to the two extremes of the gesture, one with the index finger fully extended (and hence close to the pointing gesture #9), one with the index finger flexed (close to #7 "thumbs up").

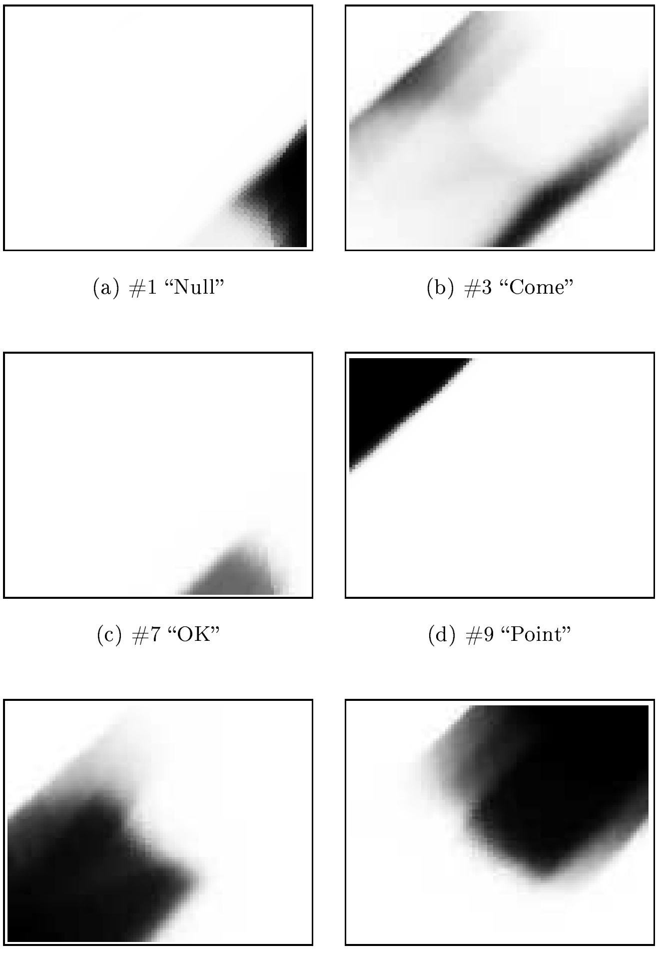
Figure 3.6 shows the activations of the relevant output units across the plane. As can be seen some decision boundaries are quite sharp, while in some areas several different classifications are possible. Most notable is the grey area between the two domains where #3 is dominant, here the unit is active but less activated than #9 or #11 (this is natural, since the gesture moves between the two extremes).

When the largest activation of the output units is plotted a network-like pattern emerges (figure 3.7). The decision boundary is marked by comparatitively uncertain classifications where several output units compete. If the gesture recognition system ignores all gesture classifications with less than 0.5 output activation the risk for random flipping between choices when the posture is changed is decreased.

3.2.2 Dependency on training time

Apparently the network converges quickly towards an optimal classifier (Figure 3.8). There are no signs of overtraining, which was expected since overtraining in this model depends more on the number of RBF units than the length of training.

Hypercolumns speed up training significantly, and it levels out already after around 500 iterations. Without hypercolumns this does not occur until after around 1000 iterations. The likely explanation is that dividing the input space into lower-dimensional subspaces makes convergence in each subspace much faster.



(a) #1 "Null" (b) #3 "Come"

(c) #7 "OK" (d) #9 "Point"

(e) #10 "Cylinder" (f) #11 "Beg"

Figure 3.6: Activations of output units across the interpolated plane. Dark = high activation. Note that some classes are more certain and strictly delineated than others, such as #9 compared to #3.

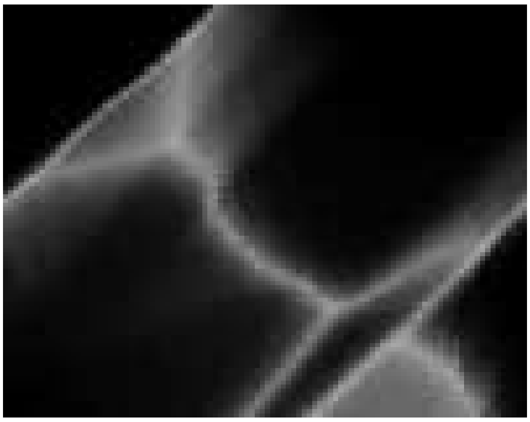


Figure 3.7: Maximal output activation for all units in the interpolated plane. The grey tone along the decision boundary corresponds to 0.35-0.5 output activation of the most active unit.

3.2.3 Number of Necessary Units

How many RBF units are needed per dimension of the input space? According to the data, even a small number can provide enough resolution to classify the postures, which both reduces training time and improves the computational performance of the network (Figure 3.9).

If very many units are used per dimension (≈ 30) overfitting occurs, and the network loses its ability to generalize to gestures made by other people or in other datasets (the ability to correctly recognize patterns belonging to the same dataset, even when not seen before, is still good at this number of units). As can be seen there is no significant overfitting for < 15 units/dimension.

In fact, it is surprisingly good even using very few units, which is due to that the Bayesian layer makes a priori estimates of the probability of the different gesture classes, and hence can make "educated guesses" based on the limited information from the RBF layer (cf. section 3.2.7).

3.2.4 Dependency on Hypercolumns

Choosing a good set of hypercolumns can improve the network considerably. Without any hypercolumns, the network categorizes around 87.5% of all patterns correctly; with a hypercolumn division the percentage rises to 93% (Figure 3.10). Placing all input dimensions in the same hypercolumn seems to be roughly as efficient as using no hypercolumns. It is interesting to note that the exact hypercolumn division doesn't matter as much as having a hypercolumn structure in the network.

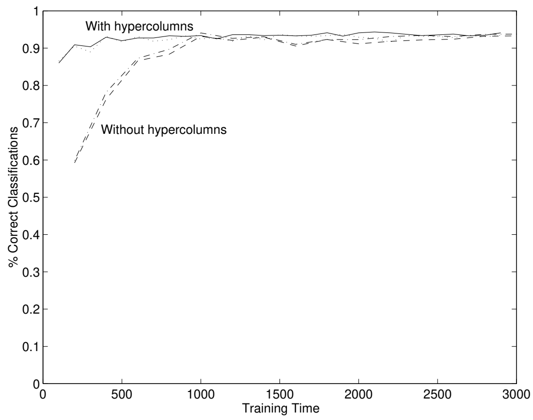


Figure 3.8: The percentage of correct classifications as function of the length of the CSL training (number of iterations of the algorithm). Dotted line: test data with hypercolumns, filled line training data with hypercolumns, dot-dashed line test data without hypercolumns and dashed line training data without hypercolumns. As can be seen, using hypercolumns significantly speed up training.

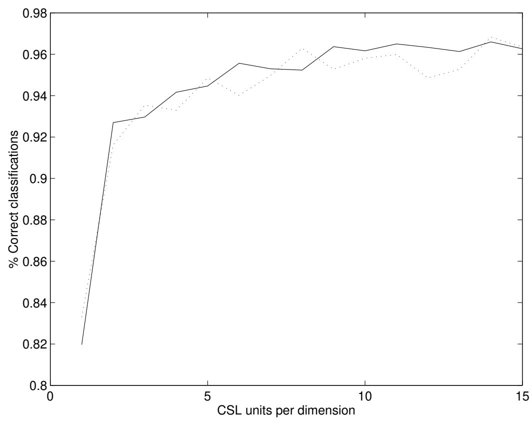


Figure 3.9: The percentage of correct classifications as function of the number of RBF units per dimension of the input space. Dotted line is test data.

3.2.5 Choosing Hypercolumns

That a good hypercolumn division is necessary is suggested by the results in previous section, which show a several percentage points increase in correct classifications by introducing hypercolumns, and section 3.2.2 which shows that hypercolumns speed up training.

Choosing a suitable set of hypercolumns/CSL input spaces is nontrivial. On one hand correlated inputs should be placed in the same hypercolumn, but the number of Bayesian units will increase exponentially with the number of strongly

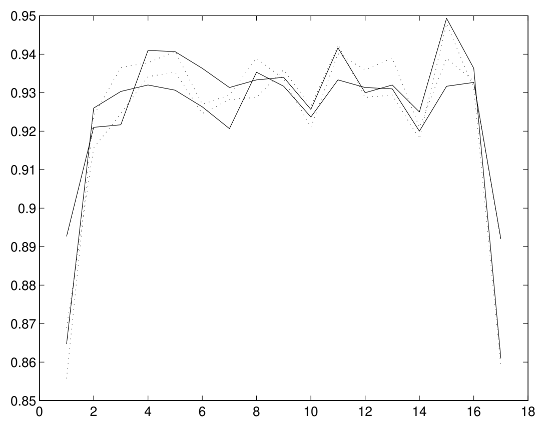


Figure 3.10: Correct classifications depending on hypercolumn structure in two trials (dotted line = test data, solid line = training data). Leftmost is the case of no hypercolumns at all, rightmost is the single hypercolumn structure, while the other data corresponds to various hypercolumn structures in no special order. The maximum performance in this trial was #15, corresponding to the structure $(0, 1, 2)(3, 4, 5)(6)(7)(8)(9)(10, 11, 12, 13, 14, 15)$

correlated inputs, and the training time for the RBF units will also increase significantly as the number of dimensions increases. One method is to calculate the mutual information between different attributes, merge the two most correlated units or columns into a higher order column, and then repeat the process a number of times to create a decorrelated hypercolumn structure [HL96].

In several of my experiments I have used hypercolumn divisions found by trial and error, guided by analysis of correlations in the input data and then manually set. The results have been positive but most likely not optimal.

Another approach which I briefly investigated was to use genetic programming to find efficient hypercolumns [HB97, Mic95, Hol75]. Strings representing hypercolumn sets were evolved using point mutation operators and cross-over, with a fitness function based on the percentage correct classifications of the test data done using a network trained with the hypercolumns.

The results were somewhat inconclusive but intriguing. Genetic programming is a general optimization method, and increased both the mean and maximum fitness as a function of the number of generations (Figure 3.11). However, the increase was relatively minor, from a mean percentage of 93.7% correct classifications using the random hypercolumns in the initial population to 95.2% performance in the final generation, a 1.5% increase. A hypercolumn structure that did well in one trial might do badly in the next due to the stochastic nature of the training process. This limits the power of genetic programming in this case, but suggests that robust hypercolumn structures have a fitness advantage before less robust structures. It should also be noted that I did not attempt to find the ideal population size and selection algorithm for the problem, which at present appears to be more an art than a science [Pha95]. Further research is needed.

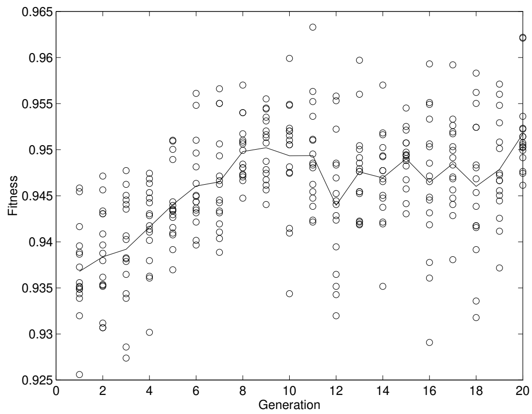


Figure 3.11: Fitness of the population of hypercolumns as a function of generation. The solid line is the generation mean fitness.

3.2.6 IIR Filter Effects

I experimented with pre-filtering the data using an IIR filter to remove noise, sending the data to the basic and the phase space network (cf. section 3.3). As can be seen in Figure 3.12, the filter does not significantly improve the basic network performance despite removing noise, and too strong filtering degrades it. The phase space network obviously benefited from filtering, since it is sensitive to high-frequency noise.

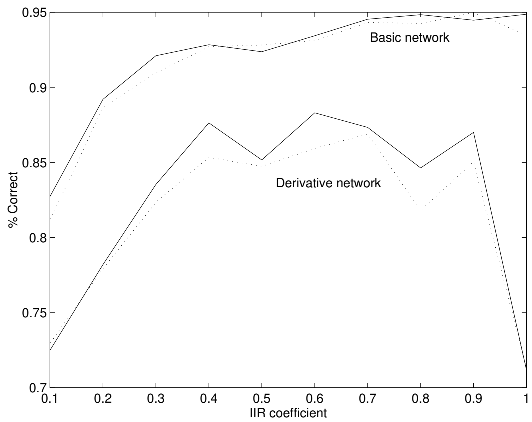


Figure 3.12: Effect of IIR pre-filtering, using basic and derivative networks. Solid line = training data, dotted line = test data.

3.2.7 Noise Resistance

I also tested the result of introducing noise into the input data (Figure 3.13). Instead of quickly breaking down at a certain noise level the network performance gradually decreases in a sigmoidal manner to roughly 50% correct classifications. At this point the network no longer tries to distinguish the different gestures, but instead defaults to the a priori most likely gesture (#1, "Null") which makes up most of the data (cf. Figure 2.9). This both shows how Bayesian neural networks strive to achieve optimal classification, and suggests that in a noisy environment the hybrid RBF-Bayesian network will be more likely to make false negative classifications (ignoring a poorly recognized gesture) than false positives (misclassifying noise as a gesture). This is clearly desirable since an over-eager gesture recognizer can be quite irritating for the user.

Different gestures are differently sensitive to noise, depending on how unique their features are and how well the network can recognize them in the noise-free signal (Figure 3.14). For example, the pointing posture with its unique straight index finger can stand much more noise than the #10 "cylinder" posture which is closer to many other postures.

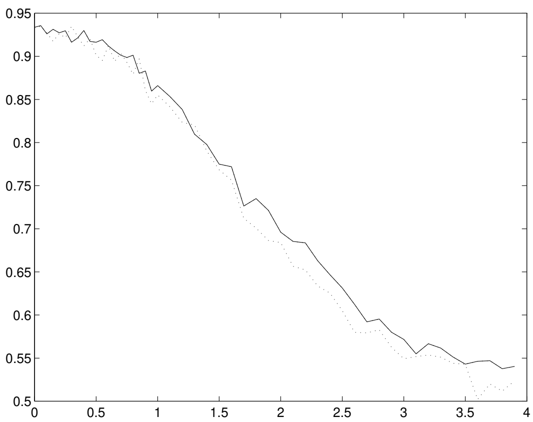


Figure 3.13: Percentage correct classifications as a function of the input noise standard deviation (rectangular probability distribution with mean zero).

3.3 Phase Space Input

A natural extension of the basic network would be to add the time derivatives of the data to the input space, creating a phase space where both configuration and movement are represented. The number of dimensions increase from 16 to 32 if all channels are used, which makes the network more cumbersome, but on the other hand the time-dependency of the network suggests that it will be better at handling the simple and perhaps also the complex gestures.

As can be seen in Figure 3.15, there are correlations in the flexion derivatives too, but the correlations between the instantaneous data and their derivatives

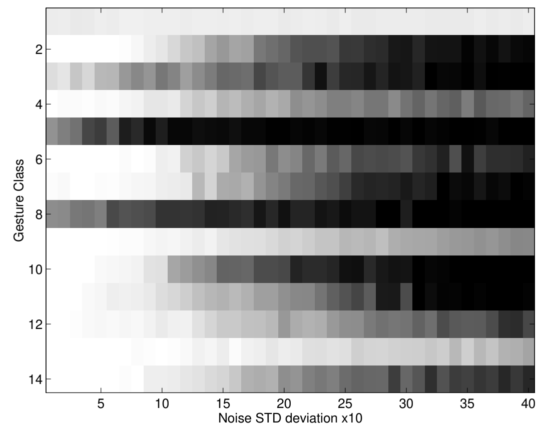


Figure 3.14: Mis-classifications of gesture classes depending on noise level. Dark shades correspond to a high number of erroneous classifications of the gesture, light shades to mostly correct classifications. Note the individual differences in noise resistance between the classes.

are weak; this suggests a natural division into subspaces.

This network handles gestures somewhat better than the previous, although it still has trouble with “Explode” (#5) and “Open” (#8), cf. Figure 3.17. “Explode” tends to be confused with “Grab” (#6) and “Open”, and “Open” seems to trigger “Explode” at the beginning. The reason is that the network still doesn't see the long time window necessary for complex gestures, but can react to typical motions. “Come” (#3) is handled significantly better, while many of the postures are sometimes misclassified. Overall, the price for better gesture recognition seems to be a slight decrease in posture certainty. This can be seen in the network response (Figure 3.16), where the amount of low-level spurious activity (which is mostly ignored by the Bayesian classification rule) is higher than in Figure 3.3. Filtration of the network output might improve its performance further. This form of network seems suitable for a mixture of postures and simple gestures, especially gestures with simple repeating or characteristic motions. It is however not enough for useful recognition of complex gestures.

One of the biggest advantages of this network is that it is more eager compared to the basic network: it reacts earlier when the user makes a gesture based on the movement itself and not just the posture (Figure 3.18). This makes recognition occur earlier and could somewhat offset lags in the system, although it also increases the risk for false positives. The speedup is on the order of one or two timesteps (≈ 0.13 s), which is significant for “spring loaded” gestures (esp. postures such as flying or grabbing) where the user desires speedy recognition.

I compared the training time needed to achieve good classification for this kind of network with the results in Figure 3.8 but found no statistically significant differences. Apparently convergence time does not depend strongly on the number of dimensions of the network, at least with a good hypercolumn structure.

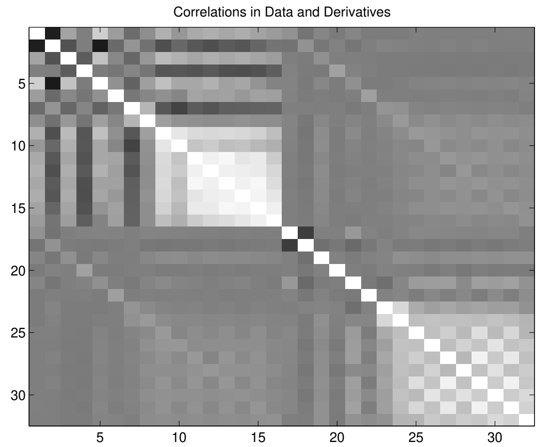


Figure 3.15: Correlations in the data and their derivatives. Dark shades correspond to negative correlations, white to positive correlations and medium grey to correlations close to zero. Channel $1, 2, 3$ = position, 4,5,6 = direction, 7–16 = finger flexion, 17–19 = position derivative, 20–22 = direction derivative, 23–32 = flexion derivatives.

3.4 Time Delay Network

Another common way of handling time in neural networks is Time Delay Networks: the input consists of the present and delayed versions of the input. This way the network is aware of the past and hopefully can detect temporal features. What has to be determined is the number k of delayed versions of the input data, the delay τ, what kind of preprocessing to use on the delays (moving averages, FIR or IIR filters are common) and a good hypercolumn structure.

At first, my results with TDNN networks were not very encouraging, due to the high number of input dimensions the network was slow to train and tended to become good at classifying complex gestures at the expense of postures. Much of this was likely due to the lack of a good hypercolumn structure in the early experiments.

However, I found a very simple hypercolumn structure which improved the performance significantly. Since the input consists of several delayed versions of the same input channels, there are strong correlations between $x_i(t)$ and $x_i(t - k\tau)$, at least for $k \cdot \tau < 10$ timesteps. It is natural to bundle together these input channels into CSL subspaces and hypercolumns, which is somewhat reminiscent of a Takens embedding[2]. Essentially the RBF units cover 16 k-dimensional input

[2]Takens observed that in general it is possible to reconstruct an attractor in state space of a dynamical system by substituting time lagged observations for higher-order derivatives (called a delay- or Takens-embedding of the data) [Tak81]. A TDNN in general is also in some sense an attempted Takens embedding, where the number of banks k represents the embedding dimension and τ the delay; cf. [LGHK96].

The main problem of embedding is to find a suitable embedding dimension; according to Taken's Embedding theorem the time-lag does not really matter, although it should not be too small (since the attractor degenerates into a space diagonal) and not too large (since the embedding then relates points of the time series which do not correlate dynamically.

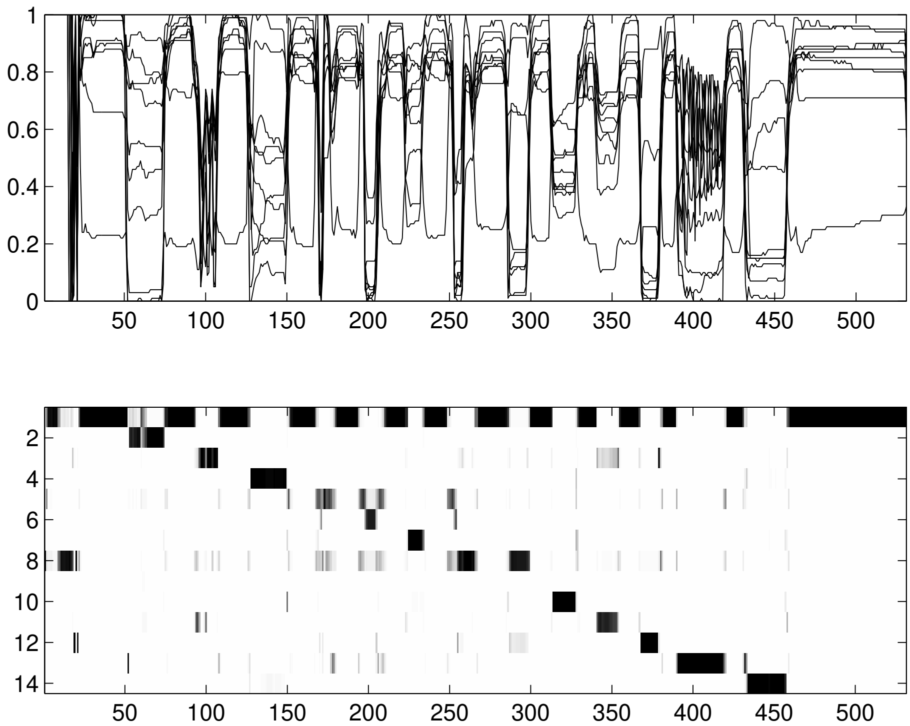


Figure 3.16: Phase space network response to sample data, analogous to Figure 3.3. Note the higher incidence of weak noise in the output units and the better classification of gesture #3, #6 and #8.

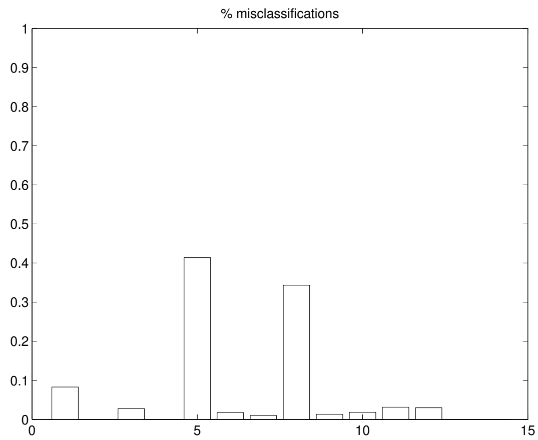


Figure 3.17: Misclassification histogram for phase space network. Mean percentage of mismatches is $\approx 6\%$ for the test set. Compare this to Figure 3.4.

Methods for determining the embedding dimension for dynamical systems are usually based on finding the box-counting dimension or geometrical analysis (the "False Nearest Neighbor

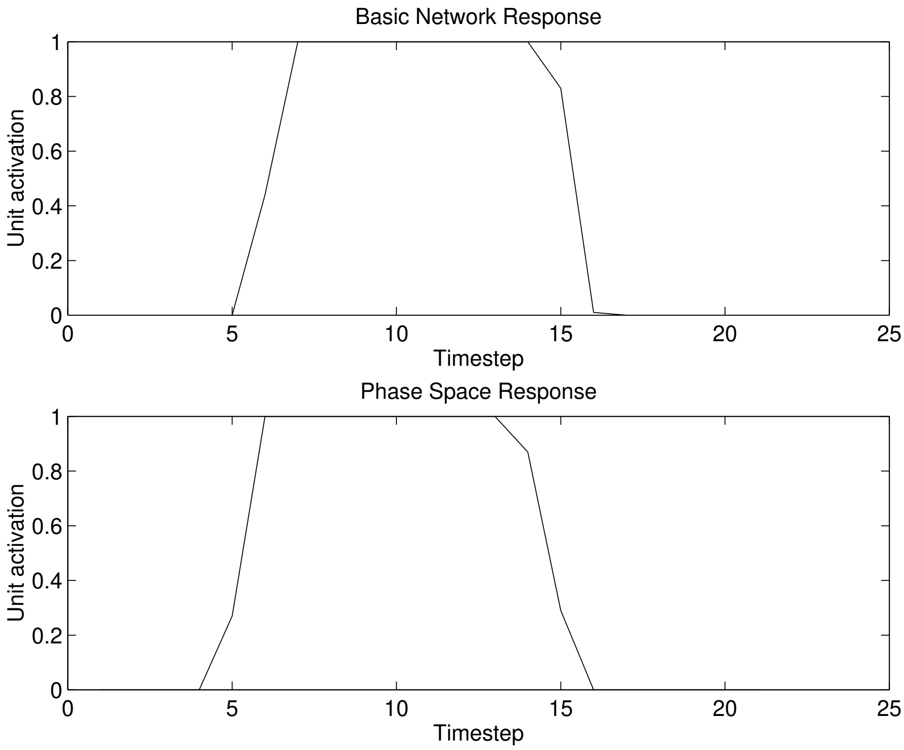


Figure 3.18: Comparison of output activation of class unit #7 "OK" in the static network (upper diagram) and the phase space network (lower diagram) when the posture is made. Note the earlier response of the phase space network, both when the gesture is formed and when it is ended.

spaces, which could be said to represent gesture states (Figure 3.19).

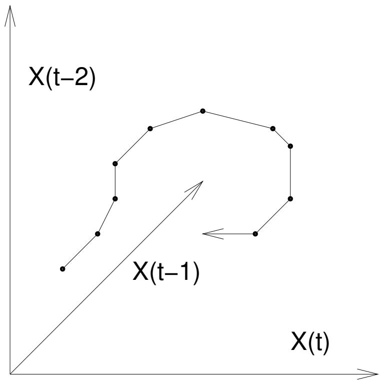


Figure 3.19: Time delay embedding of one input channel $x(t)$ into $\mathbb{R}^3$, using $k = 3, \tau = 1$.

While the TDNN version of my network makes more misclassifications in general than the basic posture recognizer or the phase space network, it is much better at recognizing complex gestures, including the troublesome #5 and #8 (Figure 3.21). The response (Figure 3.20) looks much "cleaner" than the response of the phase space network in Figure 3.16. Especially cheering is that it also does better at recognizing gesture #9 which was previously misclassified as #8.

3.4.1 Preprocessing and Other Parameters

I have attempted various forms of pre-processing for the TDNN. In general, they do not improve performance.

An alternative form of TDNN is to present moving averages of the input data of different length to the network. My experiments did not indicate that this method had any advantages; apparently the network works better if it has access to the high-frequency components of the data than if they are averaged away.

A bit surprisingly (since traditionally TDNN has had problems with variable speed signals), the time warping algorithm doesn't improve performance significantly.

Increasing the number of units per dimension on the RBF level has roughly the same effect as on the basic network.

3.4.2 Delay and number of banks

Choosing the right delay and number of banks is nontrivial, but can be guided by knowing the maximal length of gestures (around 15–20 timesteps) and the number of component movements (in a complex gesture ≈4). If the approach works, the errors should decrease for long gestures as the delay becomes comparable to their length (and increase beyond that since the time resolution

Algorithm") [KBA92].

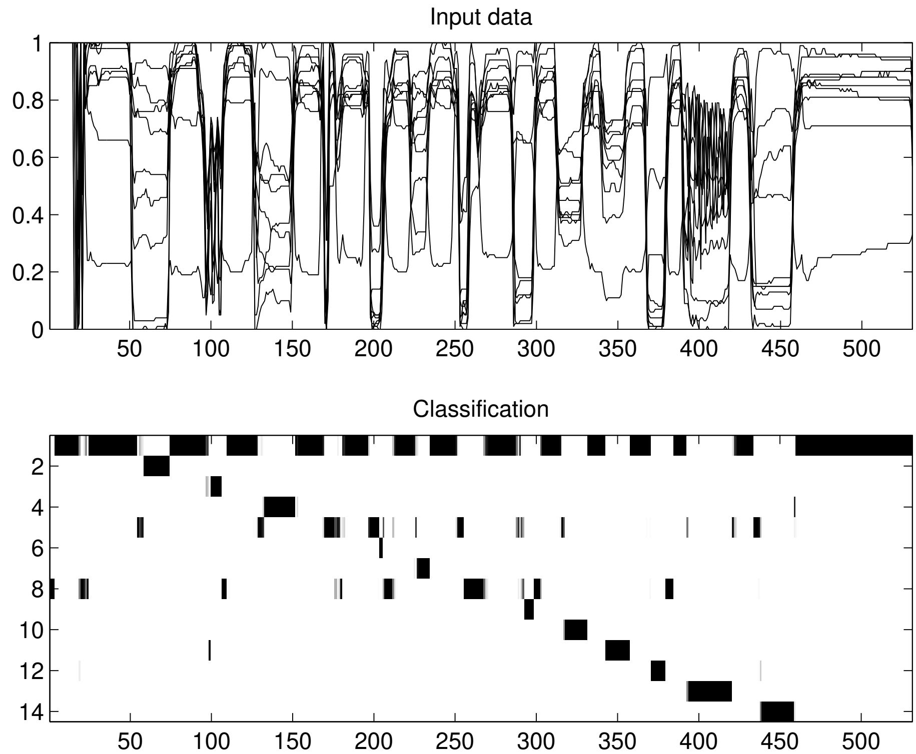


Figure 3.20: TDNN network response to sample data, $k = 4, \tau = 3$.

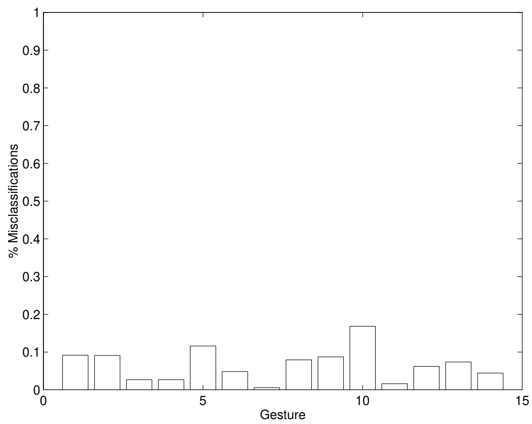


Figure 3.21: Misclassification histogram for the time delay network using the Takens-like hypercolumn structure ($k = 4$, $\tau = 7$). Mean percentage of misclassifications is $\approx 7\%$ for the test set.

becomes too coarse) and the number of banks becomes large enough to cover the constituent parts. In Figure 3.22 the results of different choices of τ and k are displayed, showing a significant decrease in mis-classifications of the troublesome gesture #8 as τ increases. However, beyond a certain point the number of mis-classifications of #5 begins to rise. An optimal choice seems to be $\tau \cdot k \approx 25$, which balances the increasing number of errors on gesture #5 against the decreasing errors on gesture #8.

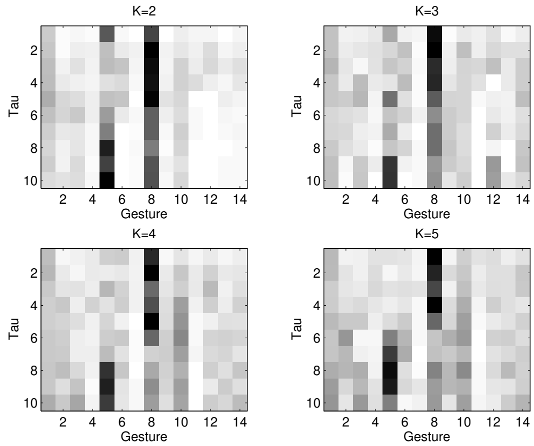


Figure 3.22: Misclassifications of different gestures as a function of delay τ and number of delay banks k. Darker shades correspond to a higher number of mis-classifications (100% black = 40% misclassifications).

It should be noted that this behavior seems to be linked to the specific hypercolumn structure used; when no hypercolumns are used at all, errors in recognizing #5 and #8 decrease with increasing τ, but instead errors in classifying #1 (false positives) increases significantly. Most likely this is an effect of the growing dimensionality of the RBF-space: the training patterns for the null gestures become more thinly spread. This is supported by the increase in errors of gesture #1 at the same value of $\tau \cdot k$ as k increases. There is a similar but weaker trend in the hypercolumn data.

3.5 Context Input

Context information simplifies recognition by modulating the probabilities of gesture classes; some gestures are only likely or allowed in some environments.

One way of achieving context sensitivity would be to make use of the second highest or third highest probability estimates from the output layer of the Bayesian network (which have so far been ignored in my studies). One simple way of using them would be to multiply the probability estimate with a weight vector dependent on the context, and then choosing the highest resultant probability. This post-processing could in principle be done by giving the Bayesian layer an extra input layer with context data, but a more efficient implementation might be a simple mask of allowed/disallowed classes.

Another solution which I have tested is to give the network some context input, in this case an extra input dimension containing simulated information about whether the virtual hand is inside an object or in free space (Figure 3.23). Certain gestures are meaningless inside an object (In this experiment pointing gestures such as #3 Come, #9 Point, #10 Cylinder and #11 Beg), others are meaningless outside objects (#5 Explode, #6 Grab, #8 Open, #13 Walk and #14 Y) and some are independent of the objects (#1 Null, #2 Bless, #3 Heavy Metal, #7 OK and #12 Thumbs). The training data consists of the ordinary glove input plus a data channel whose value corresponds to a simulated collision detection flag which is high when the gesture is likely to be done inside an object and low otherwise.

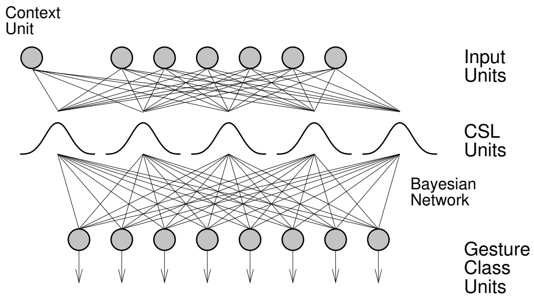


Figure 3.23: A simple context-sensitive network architecture.

At first I tested this form of context input with a static network, with noticeable improvement. The percentage of correct recognitions rose from $93.6 \pm 1.05\%$ to $95.7 \pm 1.05\%$ and especially gesture #8 was significantly better recognized (Figure 3.24). This was rather unexpected, as the gesture was both a complex gesture and belonged to the same context-class as the Grab gesture it was mainly confused with. Apparently the extra information was enough to distinguish them better.

A second test employed some gestures with random context-states, corresponding to gestures that could be made anywhere. This did not degrade the network performance.

A third test set the context input to a random value, and as the training patterns were generated put all "erroneous" patterns (object-related gestures with no objects, non-object related gestures in objects) into class #1, the null-gesture. This apparently improved recognition slightly, to $96.0 \pm 1.05\%$

In all cases the number of false positives were significantly lower than in the case with no context input, which is clearly desirable. The context input acts inhibitory on gestures that are not possible, making allowed gestures more likely.

Context input also enhanced performance in the phase space network (Figure 3.25), both in recognizing gesture #8 and many postures (such as 9–12) that were occasionally misclassified. The performance increase was from $93.3 \pm 1.05\%$ to 96.5 ± 1.05.

Adding context input to the TDNN network improved performance from $91.2 \pm 1.05\%$ to $93.8 \pm 1.05\%$ ($k = 4, \tau = 7$), Figure 3.26. The comparatively lower

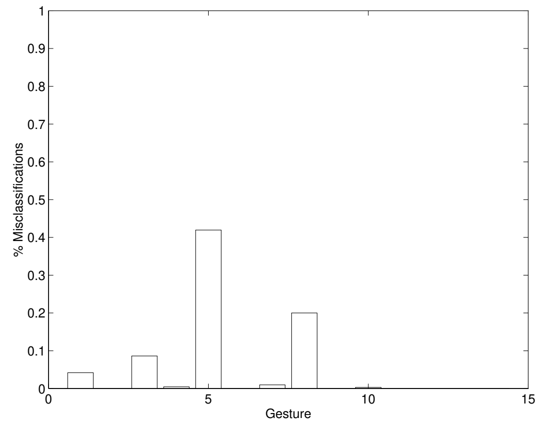


Figure 3.24: Misclassification histogram for the static network with context input. Note the significantly lower number of misclassifications of gesture #8 than in Figure 3.4.

performance increment may be due to the larger input space, the context signal simply gets "lost" in the input.

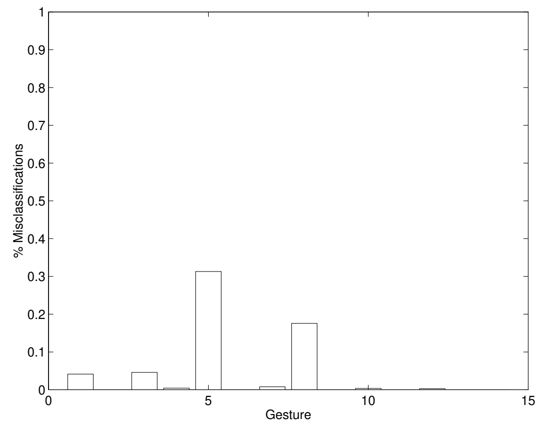


Figure 3.25: Misclassification histogram for the phase space network with context input.

3.6 Processing Demands

The recognition is quite fast, even when using basic MCSL with no optimizations for the gesture recognition task. Processing on a Sun SPARCstation 20 with 73 megabyte internal memory achieved 150 patterns/second when doing basic posture recognition and 53 patterns/second when using a 4-bank TDNN network. This processing speed is much faster than the sampling rate from the

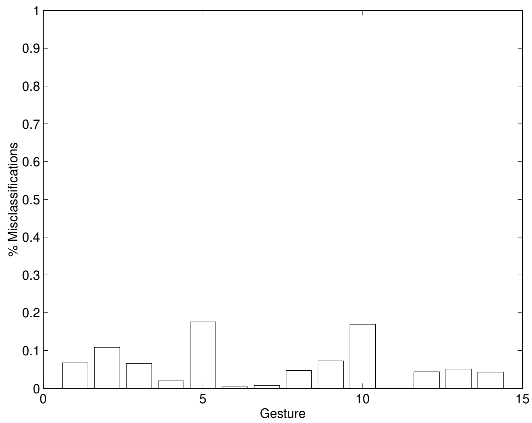


Figure 3.26: Misclassification histogram for the TDNN network with context input ($k = 4, \tau = 7$).

glove, suggesting that at least on a comparatively powerful computer even large TDNN can be used in realtime.

In comparison, a SPARCstation IPX with 35 megabyte internal memory only managed 26 patterns/second with the basic network and 10 patterns/second using the TDNN network. It is hence unlikely a full gesture recognition system could run at the same time as a virtual reality application on a smaller computer. It should however be noted that some virtual reality systems are inteded to run in a distributed fashion on several networked computers, such as the GIVEN system [VB93] and DIVE, so theoretically the different tasks could be divided between several slower computers.

Chapter 4

Discussion

> A good question is never answered. It is not a bolt to be tightened into place but a seed to be planted and to bear more seed toward the hope of greening the landscape of idea. —*John Ciardi*

It appears possible to do both posture and gesture recognition well using the RBF-Bayesian network I have used. Elaborate pre-processing appears to be less important than fitting the right kind of network to the task.

One problem is that the results of the networks are not entirely commensurable, since some are better at certain classes of gestures than others. The static network is for example very good at recognizing postures, but fails for most gestures, while the TDNN approach increases the mean number of misclassifications but can handle gestures much better.

In the case of basic posture recognition a high accuracy is possible which is to be expected. Recognition of gestures appears to require accepting a certain increase in mis-classifications of postures, at least with this method; the application will determine if this is acceptable or not. In many applications simple posture recognition works well and the addition of (complex) gestures will not contribute to the task, and hence would be unnecessary and wasteful to include. In other applications complex gestures are necessary (such as sign language recognition), and a certain amount of mistakes can be accepted in exchange for the flexibility and greater expressiveness of gestures.

Context sensitivity can circumvent many of these problems by giving the network "additional senses" with which to react to the situation. As can be seen in Figure 4.1 it improves the performance in general and in particular for certain gestures. It is quite feasible to reach less than 20% misclassifications even for complex gestures, and usually simple gestures and postures can be recognized nearly perfectly. The context model used in this network is fairly simple; more elaborate systems can be implemented in the gesture interpretation layer or in more complex network architectures. It would also be interesting to add several context inputs to the system, for example corresponding to object proximity, tool use and location.

In comparison with similar systems this model does quite well. One of the main weaknesses of this work is the comparatively small number of gesture classes

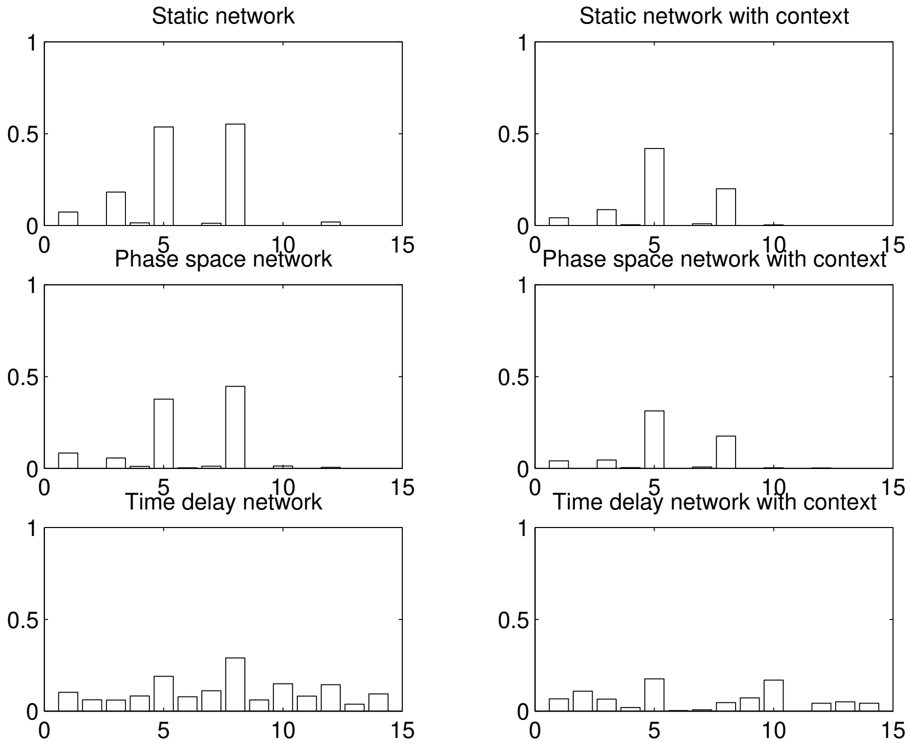


Figure 4.1: Misclassification histograms for the three different network structures with and without context input.

used in the experiments. Ideally a very large vocabulary of gestures should have been used, but due to time constraints when gathering gestures from different users and lack of disk space I was forced to use these 14[1]. A natural extension of this work would be to test the networks with more gesture classes.

While gesture recognition methods I have analyzed all perform well, the question is if they perform well enough to be useful in real life. After all, even if the system works correctly 97% of the time one out of every thirty gestures will be misinterpreted, presumably to the great annoyance of the user. This is something that has to be tested with real users and not just abstract analysis.

It should however be noted that it is likely that there exists overlap between the gesture classes; they should be regarded as fuzzy sets rather than strict categories from a gesture interaction perspective. This means that 100% recognition may not be a relevant goal; it would just correspond to total recognition of the training set and hence likely to overtraining. Instead the goal of a gesture recognition system should be to present the most likely gesture classes to the gesture interpreter, with a high degree of user-independence and dynamic resolution. Furthermore, gestures are by themselves vague and their meaning not necessarily well defined [VB93].

A natural future extension of this work could be an actual real-time implementation of the gesture recognization model, and a study of how to integrate it with gesture interpretation. Other possible extensions of this work would be to test different neural network architectures, such as a network classifying both the current posture and the most likely gesture, giving a gesture interpreter the choice between signalling a posture or a gesture. Context-sensitive gesture recognition and the use of genetic algorithms for selecting hypercolumn structures are also worth exploring further.

[1]Although as I argued in the preface, a virtual reality application is unlikely to require significantly more gestures; Volker Kuehn suggested that different contexts can be provided using 3D widgets [Wex95]

Bibliography

[Ban93] D.S. Banarse. Hand posture recognition with the neocognitron network. Master's thesis, School of Electronic Engineering and Computer Systems, University College of North Wales, Bangor, December 1993. http://www.sees.bangor.ac.uk/~dbanarse/msc/msc_thesis.ps.gz.

[BBL93] Thomas Baudel and Michel Beaudouin-Lafon. Charade: Remote control of objects using free-hand gestures. *Communications of the ACM*, 36(7):28–35, July 1993. http://www.lri.fr:80/~thomas/Docs/cacm.html.

[BBS94] Klaus Böhm, Wolfgang Broll, and Michael Sokolewicz. Dynamic gesture recognition using neural networks; a fundament for advanced interaction construction. *Proceedings of the SPIE*, 2177:336–346, 1994.

[BE92] R. Beale and A. Edwards. Recognising postures and gestures using neural networks. In R. Beale and J. Finlay, editors, *Neural Networks and Pattern Recognition in Human Computer Interaction*. Ellis Horwood, 1992.

[Ber94] Michael R. Berthold. A time delay radial basis function network for phoneme recognition. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 7, pages 4470–4473, 1994.

[BHV92] Klaus Böhm, Wolfgang Hübner, and Kaisa Väänänen. Given: Gesture driven interactions in virtual environments — a toolkit approach to 3d interactions. In *Interfaces to Real and Virtual Worlds-conference*, March 1992. http://www.zgdv.de/www/zgdv-vc/publications/online/given1.ps.gz.

[BMHW93] C. Bregler, S. Manke, H. Hild, and A. Waibel. Improving connected letter recognition by lipreading. In *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 1993. http://www.is.cs.cmu.edu/papers/multimodal/93.icassp.chris.ps.gz.

[Bro89] Martin Brooks. The DataGlove as a man-machine interface for robotics. In *The Second IARP Workshop on Medical and Healthcare Robotics*, pages 213–225, Newcastle upon Tyne, UK, September 5–7 1989.

[Bry93] Steve Bryson, editor. *Implementing Virtual Reality*, number 43 in SIGGRAPH Course Notes, August 1993.

[DS77] Thomas A. DeFanti and Daniel J. Sandin. Final report to the National Endowment of the Arts. Technical Report US NEA R60-34-163, University of Illinois at Chicago Circle, Chicago, IL, 1977.

[EP96] Neil R. Euliano and Jose C. Principe. Spatio-temporal self-organizing feature maps. In *Proceedings of the ICNN*, volume 4, pages 1900–1905, 1996. `http://www.cnel.ufl.edu/~neil/tempkoh/temporal.html`.

[Fel94] S. Sidney Fels. *Glove-Talk II: Mapping Hand Gestures to Speech Using Neural Networks — An Approach to Building Adaptive Interfaces.* PhD thesis, University of Toronto, January 1994. `http://psy.uq.oz.au/~michaeln/crg/abstracts/fels.glove.html`.

[FS91] James A. Freeman and David M. Skapura. *Neural Networks: Algorithms, Applications and Programming Techniques.* Addison-Wesley, 1991.

[FS94] Masaaki Fukumoto and Yamhito Suenaga. "fingering": A full-time wearable interface. In *Proceedings of CHI 94*, volume 2, pages 81–82, 1994.

[Gon94] Ruben Gonzalez. *Video Coding for Hypermedia Based on a Computer Graphic Model.* PhD thesis, University of Technology, Sydney, 1994. `http://jupiter.tsrc.uow.edu.au/~ruben/phd/phd.html`.

[Gri83] Gary J. Grimes. Digital data entry glove interface device. US Patent 4,414,537, November 1983.

[Hag96] Olof Hagsand. Interactive multiuser VEs in the DIVE system. *IEEE Multimedia*, 3(1):30–39, 1996.

[HB97] Joerg Heitkoetter and David Beasley. The hitch-hiker's guide to evolutionary computation: comp.ai.genetic FAQ, 1997. `http://www.cs.colostate.edu/~dzubera/FAQs/genetic.faq`.

[HDGS83] F. R. A. Hopgood, D. A. Duce, J. R. Gallop, and D. C Sutcliffe. *Introduction to the Graphical Kernel System (G.K.S.).* Number 19 in A.P.I.C Studies in Data Processing. Academic Press, London, England, 1983.

[HL93] Anders Holst and Anders Lansner. A bayesian neural network model with extensions. Technical Report TRITA-NA-P9325, Royal Inst. of Technology, Sweden, 1993. `ftp://ftp.nada.kth.se/pub/documents/SANS/reports/ps/TRITA-NA-P9325.ps.Z`.

[HL96] Anders Holst and Anders Lansner. A higher order bayesian neural network for classification and diagnosis. In A. Gammerman, editor, *Computational Learning and Probabilistic Reasoning*, chapter 12.

John Wiley & Sons, Ltd, 1996. Proc. of ADT, London, England, April 3–5, 1995.

[Hol75] J.H. Holland. *Adaption in Natural and Artificial Systems.* MIT Press, 1975.

[HSM94] Chris Hand, Ian Sexton, and Michael Mullan. A linguistic approach to the recognition of hand gestures. Presented at the *Designing Future Interaction* conference (Ergonomics Society/IEE), University of Warwick, UK, April 1994. `http://www.cms.dmu.ac.uk/~cph/Publications/DFI94/gestures.html`.

[HT89] Jiawei Hong and Xiaonan Tan. Calibrating the VPL DataGlove for teleoperating the Utah/MIT hand. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 1752–1757, 1989.

[JA92] Kai-Mikael Jää-Aro. Tankathon — a process plant simulation. Technical Report TRITA-NA-P9207, MultiG-6, NADA, 1992. `ftp://ftp.nada.kth.se/IPLab/TechReports/IPLab-52.ps.Z`.

[JA96] Kai-Mikael Jää-Aro. *Three-dimensional Interaction and Presentation for Computer-Supported Cooperative Work.* Licentiate thesis, Royal Institute of Technology, 1996.

[JFH94] Rick Jacoby, Mark Ferneau, and Jim Humphries. Gestural interaction in a virtual environment. *Proceedings of the SPIE*, 2177:355–363, 1994.

[Kad95] Mohammed Waleed Kadous. Grasp: Recognition of Australian sign language using instrumented gloves. Master's thesis, University of New South Wales, october 1995. `http://www.cse.unsw.edu.au/~waleed/thesis.html`.

[Kal93] Roy S. Kalawsky. *The science of virtual reality and virtual environments: a technical, scientific and engineering reference on virtual environments.* Addison-Wesley, 1993.

[KBA92] Matthew B. Kennel, Reggie Brown, and Henry D. I. Abarbanel. Determining embedding dimension for phase–space reconstruction using a geometrical construction. *Physical Review A*, 45(6):3403–3411, 1992.

[Keu95] Johannes Keukelaar. External event handling in DIVE, a virtual reality system. Technical Report TRITA-NA-E9531, NADA, 1995. `ftp://ftp.nada.kth.se/IPLab/TechReports/IPLab-94.ps.Z`.

[Koh84] Teuvo Kohonen. *Self-Organization and Associative Memory*, volume 8 of *Springer Series in Information Sciences.* Springer-Verlag, 1984.

[Kru91] Myron W. Krueger. *Artificial Reality II.* Addison-Wesley, Reading, MA, 1991.

[Kru93] Myron W. Krueger. The emperor's new realities. *Virtual Reality World*, 1(3/4):18–33, November/December 1993.

[LBG80] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1), 1980.

[LE89] Anders Lansner and Örjan Ekeberg. A one-layer feedback, artificial neural network with a Bayesian learning rule. *Int. J. Neural Systems*, 1(1):77–87, 1989.

[LGHK96] Tsungnam Lin, C. Lee Giles, Bill G. Horne, and S.Y. Kung. A delay damage model selection algorithm for NARX neural networks. Technical Report CS-TR-3707, University of Maryland Institute for Advanced Computer Studies Dept. of Computer Science, Univ. of Maryland NEC Research Institute Dept. of Electrical Engineering, Princeton University, December 1996. `ftp://ftp.cs.umd.edu/pub/papers/papers/3707/3707.ps`.

[LH96] Anders Lansner and Anders Holst. A higher order Bayesian neural network with spiking units. *Int. J. Neural Systems*, 7(2):115–128, 1996.

[LX96] Christopher Lee and Yangsheng Xu. Online, interactive learning of gestures for human/robot interfaces. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 2982–2987, 1996. `http://www-cgi.cs.cmu.edu/afs/cs/project/space/www/hmm/hmm-icra96.ps.gz`.

[Man97] Steve Mann. Wearable computing: A first step toward personal imaging. *Computer*, 30, February 1997. `http://www.computer.org/pubs/computer/1997/r2025.htm`.

[McN92] D. McNeill. *Hand and mind: what gestures reveal about thought.* Chicago, University of Chicago Press, 1992.

[Mic95] Zbigniew Michalewicz. A perspective on evolutionary computation. In Xin Yao, editor, *Progress in Evolutionary Computation*, volume 956 of *Lecture Notes in Artificial Intelligence*, pages 73–89, Berlin, 1995. Springer-Verlag.

[MT91] K. Murakami and H. Taguchi. Gesture recognition using recurrent neural networks. In *CHI '91: Human Factors in Computing Systems: Reaching Through Technology*, pages 237–241. SIGCHI, April/May 1991.

[Mul96] Axel Mulder. Hand gestures for HCI. Technical Report 96-1, School of Kinesiology, Simon Fraser University, February 1996. `http://fas.sfu.ca/cs/people/ResearchStaff/amulder/personal/vmi/HCI-gestures.htm`.

[New94] Gregory B. Newby. Gesture recognition using statistical similarity. *Presence: Teleoperators and Virtual Environments*, 3(3):236–244, summer 1994. `http://www.cms.dmu.ac.uk/People/cph/VRbib/Gesture/newby93.txt`.

[Pha95] Q. Tuan Pham. Competitve evolution: A natural approach to operator selection. In Xin Yao, editor, *Progress in Evolutionary Computation*, volume 956 of *Lecture Notes in Artificial Intelligence*, pages 49–60, Berlin, 1995. Springer-Verlag.

[Pic95] R. W. Picard. Affective computing. Technical Report 321, MIT Media Laboratory Vision and Modelling Section, November 1995. `ftp://whitechapel.media.mit.edu/pub/tech-reports/TR-321.ps.Z`.

[PKBL83] H. Poizner, E.S. Klima, U. Bellugi, and R.B Livingston. Motion analysis of grammatical processes in visual-gestural language. In *Proc. ACM SIGGRAPH/SIGART Workshop on Motion*, pages 148–171, April 1983.

[PS89] Lucy Pao and Thomas H. Speeter. Transformation of human hand positions for robotic hand control. In *Proc. IEEE International Conference on Robotics and Automation*, volume 3, pages 1758–1763, 1989.

[Qui92] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

[Rub91] Dean Rubine. Specifying gestures by example. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 329–337, July 1991.

[Shn82] B. Shneiderman. The future of interactive systems and the emergence of direct manipulation. *Behaviour and Information Technology*, 1(3):237–256, 1982.

[Smi96] Joshua R. Smith. Field mice: Extracting hand geometry from electric field measurements. *IBM Systems Journal*, 35:587–609, 1996. `http://isj.www.media.mit.edu/projects/isj/SectionE/587.htm`.

[SP90] Thad Sterner and Alex Pentland. Real-time American sign language recognition from video using hidden Markov models. Technical Report 375, MIT Media Laboratory Perceptual Computing Section, 1990. `ftp://whitechapel.media.mit.edu/pub/tech-reports/TR-375.ps.Z`.

[Stu92] David J. Sturman. *Whole-hand Input*. PhD thesis, MIT, February 1992. `ftp://ftp.media.mit.edu/pub/sturman/WholeHandInput/thesis.pix.ps.Z`.

[SZP89] David Sturman, David Zeltzer, and Steven Pieper. Hands-on interaction with virtual environments. In *Proc. UIST '89: ACM SIGGRAPH/SIGCHI Symposium on User Interface Software and Technology*, pages 19–24, November 1989.

[Tak81] F. Takens. Detecting strange attractors in turbulence. In D.A. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 366–381, Berlin, 1981. Springer-Verlag.

[UN94] Naonori Ueda and Ryohei Nakano. A new competitive learning approach based on an equidistortion principle for designing optimal vector quantizers. *Neural Networks*, 7(8):1211–1227, 1994.

[VB93] Kaisa Väänänen and Klaus Böhm. Gesture driven interaction as a human factor in virtual environments — an approach with neural networks. In R. Earnshaw, M. Gigante, and H. Jones, editors, *Virtual Reality Systems*, pages 93–106. Academic Press, 1993. ISBN 0-12-227748-1 `http://www.zgdv.de/www/zgdv-vc/publications/online/given2.ps.gz`.

[VPL89] VPL Research Inc. *Dataglove Model 2 Operation Manual*, January 1989.

[Wat93] Richard Watson. A survey of gesture recognition tecniques. Technical Report TCD-CS-93-11, Department of Computer Science, Trinity College, Dublin, July 1993. `ftp://ftp.cs.tcd.ie/pub/tcd/tech-reports/reports.93/TCD-CS-93-11.ps.gz`.

[WB95] Andrew Wilson and Aaron Bobick. Using configuration states for the representation and recognition of gesture. Technical Report 308, MIT Media Laboratory Vision and Modelling Section, 1995. `ftp://whitechapel.media.mit.edu/pub/tech-reports/TR-308.ps.Z`.

[Wex94] Alan Wexelblat. Natural gesture in virtual environments. In Gurminder Singh, Steven K. Feiner, and Daniel Thalmann, editors, *Virtual Reality Software & Technology*, pages 5–16. ACM & ISS, World Scientific, 1994.

[Wex95] Alan Wexelblat. Summary of CHI'95 gesture at the user interface workshop, November 1995. `http://wex.www.media.mit.edu/people/wex/CHI95-workshop-writeup.html`.

[WG87] David Weimer and S.K. Ganapathy. Interaction techniques using hand tracking and speech recognition. In Meera Blattner and Roger Dannenberg, editors, *Multimedia Interface Design*, Frontier Series, pages 109–126. Addison-Wesley, 1987.

[WHH+89] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang. Phoneme recognition using time delay neural networks. *IEEE Transactions on Accoustics, Speech and Signal Processing*, 37(3):328–339, 1989.

[YW96] Nam Yanghee and KwangYeon Wohn. Recognition of space-time hand gestures using hidden markov model. In *ACM Symposium on Virtual Reality Software and Technology*, pages 51–58, July 1996. `http://dangun.kaist.ac.kr/~nyh/camera1.ps`.

[Zim96] Thomas G. Zimmerman. Personal area networks (PAN): Near-field intra-body communication. *IBM Systems Journal*, 35:609–618, 1996. `http://isj.www.media.mit.edu/projects/isj/SectionE/609.htm`.

[ZLB+87] Thomas G. Zimmerman, Jaron Lanier, Chuck Blanchard, Steve Bryson, and Young Harvill. A hand gesture interface device. In *CHI + GI '87*, pages 189–192. SIGCHI, 1987.

All WWW URLs current as of April 21, 1997.