uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

u Ottawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Thierry Métais

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Dynamic Gesture Interface

TITRE DE LA THÈSE / TITLE OF THESIS

Nicolas D. Georganas

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Rafik Goubran

Emil M. Petriu

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

A Dynamic Gesture Interface

by

Thierry Métais

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the
degree of
Master of Science
in
Computer Science.

The Ottawa-Carleton Institute for
Computer science
School of Information Technology and Engineering
University of Ottawa

Your file Votre référence
ISBN: 0-494-11353-7
Our file Notre référence
ISBN: 0-494-11353-7

Abstract

Technological advancements during the last years made three dimensional virtual environments ubiquious computer applications. The traditional input devices such as keyboards and mice have reached their limits for they are no longer intuitive to manage 3 dimensional interactions. During the last decades, digital gloves were invented in an attempt to capture hand gestures. The work presented in this thesis is about recognizing hand shape based gestures for virtual environment navigation. We addressed this issue in two steps: first, we look for gesture representations that would allow a good discrimation between gestures; second, we have examined three strategies namely template matching, neural network and hidden Markov models to classify inputs suitably formatted. Experiments on classifying gesture samples with the three methods seem encouraging (more than 90 percent recognition), but when looking at real time gestural sequencee we conclude that only the Viterbi and neural network approaches should be used for a gesture interface.

Acknowledgements

Foremost, I would like to thank Dr Georganas for having given me the opportunity of studying the interesting and challenging area of gesture recognition, for his great patience and supervision during these years. I am also obliged to all people of the DISCOVER laboratory, particularly to Francois Malric and Mojtaba Hosseini, for all the valuable advices and support they gave me.

List of Figures

List of Tables

Contents

List of Abbreviations

GR	*Gesture Recognition*
VE	*Virtual Environment*
VR	*Virtual Reality*
DCU	*Device Configuration Utility*
ASL	*American Sign Language*
DG	*Dynamic Gesture*
EP	*Elementary Postures*
VQ	*Vector Quantization*
TFP	*Temporal Feature Problem*
LBG	*Linde-Buso-Gray*
HMM	*Hidden Markov Model*
NN	*Neural Network*
DTW	*Dynamic Time Warping*
VRML	*Virtual Reality Markup Language*
API	*Application Programming Interface*
OO	*Object Oriented*
JNI	*Java Native Interface*
GUI	*Graphical User Interface*
dll	*dynamic link library*

Chapter 1

Introduction

1.1 Generalities

Human-machine interfaces' evolution is tightly linked to the development of new applications. Indeed, at the dawn of the computing era, computers such as ENIAC were only employed to solve computationally intensive problems such as code breaking. All interactions between them and their users were limited to the problem instructions, which were transmitted by *punch cards*. Later on, computer uses were extended to office work: to support typing *keyboards* were introduced. However, with the appearance of 2D interfaces like windows, keyboards use lost their intuitiveness. *Mice* and *graphic pens*, much more natural in their use took over the keyboard directional pad.

Lately, due to the increase of computational power, 3D interfaces figuring virtual environments (**VE**) have become common, especially in electronic games. The unveiling of a third dimension triggered the invention of new hardware such as *joysticks*. Although joysticks are appropriate for flight simulations, their use for grabbing and interacting with more general VEs is less natural.

The recent technological progress has allowed us to capture and track people's hands. The incentive of detecting gestures to interact with some computer application is straightforward, for hands are an integral part of our human communication and can fully use the three dimensional degrees of freedoms to express spatial motions.

1.2 Problem statement

The work described in this thesis intends to create a dynamic gesture interface. In traditional user-computer interfaces, we can discern two parts:

- A *driver*, which is a program in charge of processing low-level generated signals from the user-input hardware into higher-level representations. In our case the primitive signals are a set of digital values generated by a digital glove auxiliary computer. The expected high level representations of these signals are gesture identifications.

- A *manager*, a software built on top of the driver, whose purpose is to make the link between the end user application and the signals generated by the driver. For example, in the case of keyboard, this software can configure additional parameters such as *repeat delay*, *repeat rate*, *cursor blink rate*, among others.

1.3 Proposed solution

In this thesis, we focused chiefly on the *driver* aspect of a gesture recognition interface, touching only briefy the *manager* part in charge of the semantic interpretation of gestures. Consequently, we aim at creating a recognition framework permitting to detect gestures sufficiently reliably so that gesture interaction could be supported.

1.4 Contributions

As a result of the work mentionned in this thesis, we have:

- developed a framework for detecting specified gestures, which can be easily used by Java or C++ user applications to incorporate gestural interactions.

- developed a file system browsing application using the previous framework. This application was demonstrated during the *1st Annual Scientific Conference of the LORNET network* [3].

- presented a paper intitled *A Glove Gesture Interface* [12] during the *22nd Biennial Symposium on Communications* at Queen's university.

1.5 Structure of the thesis

After this introductory chapter, the thesis pursues as follows:

- In chapter 2, we set some terminology about gestures and review some methods employed to recognize gestures.
- Chapters 3 and 4 successively present the two main recognition stages, namely the gesture representation and the gesture classification. The former delves in the issue of analyzing the data streams generated by the input devices to elaborate gesture models while the latter presents three possible classification strategies.
- Chapter 5 describes the implementation of the developed framework.
- Following the previous theoretical chapters, chapter 5 contains results from gesture recognition experiments based on the different models we investigated.
- In the last chapter, we summarize and draw some conclusions on the work presented.

Chapter 2

Background Knowledge

In this chapter, we first introduce the user to the input devices we use to capture gestures. Then, we pursue on presenting the challenge of detecting gestures, positioning the problem with respect to more classical recognition problems. Later, we precise the notion of gesture, that we will focus on. Finally, we touch on two implementation API that were used during the developement.

2.1 Material

To capture gestures, we have at our disposal a Virtual Technologies *CyberGlove*™ [1] (figure 2.1) and an Ascension™ tracker [2] (figure 2.3).

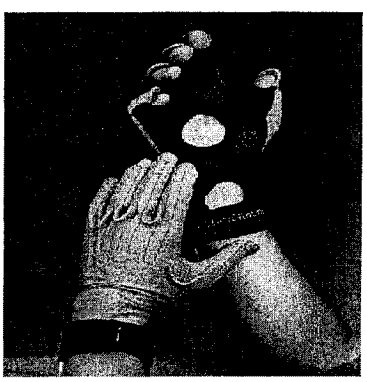

Figure 2.1: The CyberGlove™

Glove

The glove has 22 flexion sensors distributed on the finger joints (outer , middle and inner joints), on the abdduction joints (4) and on the wrist (palm arch, wrist flexion and abduction). Thanks to these sensors, digital values are generated, reflecting the current joint angles. Immersion^{TM}, the company that has created the CyberGlove^{TM}, has developed a Device Configuration Utility (**DCU**), which allows us to modulate the joint signals by adjusting the *offset* and *gain* parameters. Although both parameters are important when displaying a visual model of the hand, we set the gains of the different signals at maximum, totally ignoring their offset parameters. Indeed the greater gains are, the more discrimination they will offer between two joint angles. Once the DCU parameters set, modifying them will deprive previous glove data recordings from any meaningful interpretation. Therefore, to ensure data consistency from one experiment to another, we let the DCU parameters **constant for all experiments**.

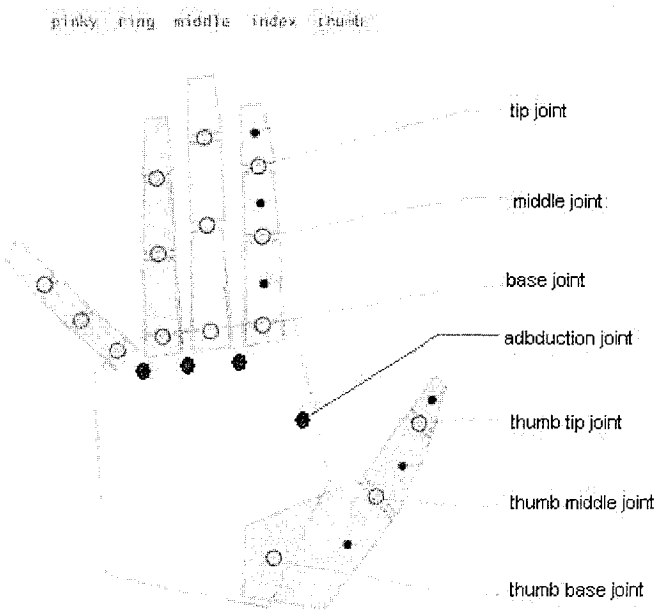

Figure 2.2: Finger joints

Tracker

On the other hand, the tracker is used for obtaining the hand spatial movements. This input hardware is composed of an *emitter* and a *bird*. The emitter is a cubic box creating a magnetic field within a range of 3 feet and a bird is a magnetic sensor. Once bird signals are processed, the bird's location and orientation with respect of the emitter referential are available. One inherent issue associated with the tracker is the mobility of its emitter, which causes inconsistency between data recorded during different experiments. Indeed, any rotation or translation affecting the emitter, immediately changes the referential attached to it. One solution we envisioned was creating an independent **user referential** ,which should be defined at the beginning of each experiment. However the definition of such referential proved to be too subjective (*front*, *right* directions, *user center*) to give any exploitable data. Additionnaly, because of the up-down symmetry of the magnetic field, generated data are subject to sign inversion, whenever the bird changes hemisphere.

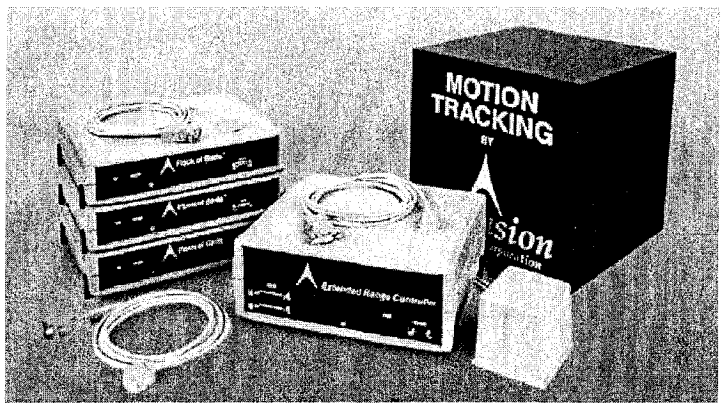

Figure 2.3: The Ascension^{TM} tracker

2.2 Gestures

The word *gesture* has many meanings, as shows its Merriam-Webster^{TM} definition:

1. a movement usually of the body or limbs that expresses or emphasizes an idea, sentiment, or attitude.
2. the use of motions of the limbs or body as a means of expression.
3. something said or done by way of formality or courtesy, as a symbol or token, or for its effect on the attitudes of others: *a political* gesture *to draw popular support – V. L. Parrington.*

For theoretical and practical reasons, we decided to restrict ourself to **single hand, hand shape based** gestures, which can be apprehended by observing the variations in the fingers configurations independently of the hand location and orientation. Moving signs will not be differenciated from their still versions.

In the next chapter, examining the spectra of the signals generated from the digital glove capture, we can observe than the hand configurations can be hold as constant over sufficiently short time intervals. We termed these instantaneous hand shapes as *elementary* postures, which can be viewed as photographs of the hand at specific times. The hand temporal behavior can thus be identified as a sequence of elementary postures.

If sucessive elementary postures are identical over a important period of time, we can then assume that the user did purposedly hold his hand poised. In that case, we consider that a *static gesture*, sign, posture has been executed. Certain sequences of various elementary postures are conscientiously made: such sequences correspond to *dynamic gestures.* When the context does not allow any misinterpretation, we will refer to dynamic gestures as *gestures* by opposition to signs.

The hand shape based gesture class may appear too restrictive for some people. Yet we judge it sufficiently large to be interesting and point at the American Sign Language (**ASL**) alphabet as a proof. Indeed all postures displayed on figure 2.4 fall in that class, with exception of the letters J and Z, that require additional spatial motion information.

Gesture recognition problems

The difficulty when attempting to recognize gestures arises from different facts, among which are:

- A gesture is an abstract fuzzy identity: even an apparent clear sign such as *pointing* may be recognized through different hand configurations, among

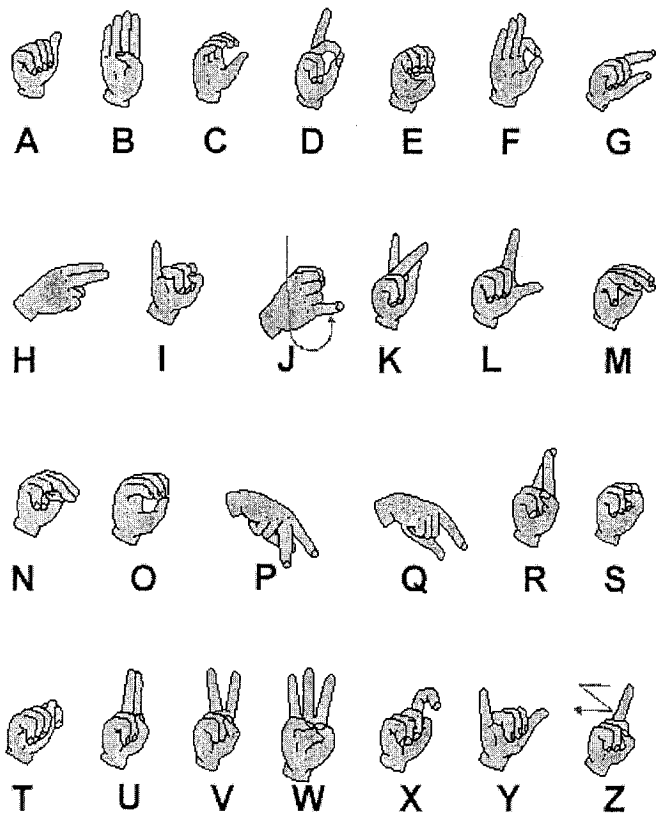

Figure 2.4: The ASL language

which the fingers configuration sightly differs. This lack of precision is caused by the semantic dimension of gestures. An obvious example of the semantic importance is the *OK* gesture, which can be done in two ways: with a thumb up or doing a zero with the thumb and the index. Dynamic gestures are even more troublesome to characterize, since their beginnings and endings are most often subjectively defined.

- Gestures' variabilities because of variations in execution speed complicates the recognition task. Although this issue is intuitive when considering different users, it can not be neglected between gesture samples from a single person. Indeed emotions directly influence one's way of performing a gesture.

- When trying to recognize gestures performed in real time, people face the *articulation* or *chaining* issue. On top of the segmentation challenge, some pair of gestures may be merged together (chained): the former's end is integrated to the latter's beginning.

Semantic considerations

As we were mainly preoccupied by the issue of gesture recognition, we did not particularly pay attention to the semantic content of gestures. However, during the implementation of some gesture based application to navigate in a directory file system, gesture signification aspects could no longer be ignored. Indeed when gestures are used for communication purpose, Krauss, Chen and Gottesman separate them into 3 main kinds [15]:

- *Symbolic* gestures: they can be used without any accompanying speech, since bearing a precise meaning. Obvious examples of symbolic gestures are provided by insults done by motioning.

- *Deictic* gestures: this category is used to refer to some contextual entities, such as location (*here, there*), or objecs (*this one, that one*). The first instance of this sort of gesture coming to mind is of course the *pointing* gesture, which is a pilar of any interaction.

- *Motor* gestures, which unlikely previous types do not support any specific sense but occur during the speech with the incentive of clarifying the

locutor's ideas. To illustrate this class of gestures we can think of politicians balancing hands alternatively to each side when discussing matters organized in different classes (*on the one hand ... on the other hand*).

2.3 Recognizing gestures

Gesture recognition (**GR**) is an particular instance of the pattern recognition problem. Indeed, in that context the patterns to discern are *gestures*. GR has been researched for several decades with the work of Thomas O. Ellis dating back to 1967 [13]. At that time, Ellis and his collegues developed a framework to recognize gestures sketched by a pen on a tablet (the *RAND* tablet invented in 1964, see figure 2.5).

Figure 2.5: RAND tablet used to recognize pen-based gestures

In order to recognize gestures, we must take the following characteristics into consideration:

- *multidimensionnality*: with most capture devices, hand gestures are characterized by more than one signals. For instance, when using a camera we have to analyze a series of pictures (bidimensionnal data) and, if we use a digital glove such as the CyberGlove^{TM}, we obtain as many signals as joints.

- *temporal aspect*: *gestures* contain a definite temporal aspect. Even hand signs: the hand has to be poised a *certain* time in a fixed configuration.

- *articulation phenomena*: when we execute some gestures in real time, there is no conspicuous separation among them in general. Looking at figure 2.6, we can spot 3 different phases possibly corresponding to distincts gestures. However the transitions between them are not clean, complicating the segmentation of input streams into different phases.

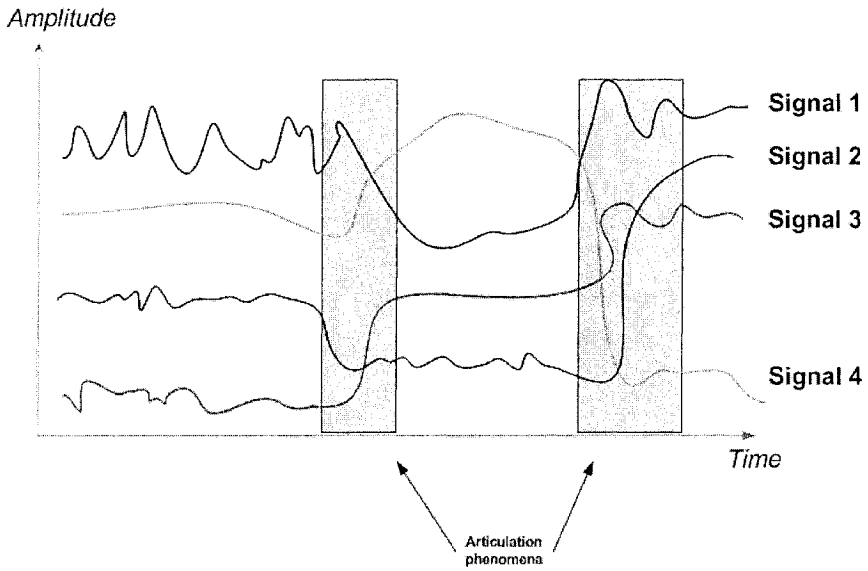

Figure 2.6: Multidimensionality, temporal , articulation aspects

These properties are not particular to the GR area, so more traditional recognition problems as *cursive writing* or *speech* recognition may be great sources of inspiration.

2.3.1 Recognition canvas

Pattern recognition solutions can usually be broken down to 3 main stages

- The **retrieval** and **preprocessing** of patterns' data.
  There are several possibilities available to observe a user's hand which were inventoried by LaViola Jr [23]. Two main hand capture paradigms consist of using a video camera or wearing a digital glove. Although the vision based capture permits a greater freedom in the motions because no

external constraints are imposed on the user's hand, we preferred using the cyberglove, which gives us direct numerical information about the fingers (compared to hand images). Moreover, an important part of the signals preprocessing has been embedded in the cyberglove auxiliary computer: it filters high frequencies, too high to be generated by human motions, with a low-pass filter of corner frequency 30 Hz. Our preprocessing work is thus lighter, focusing chiefly on the tracker signals.

- The **representation** of the pattern.
  As numerical signals coming from the hand capture have been formatted to be exploitable, we have to analyze them in order to clearly highlight each pattern's own characteristics. These characteristics are referred in the literature as *features*. The feature extraction stage is twofold: first we have to unearth some gesture aspects and then select some features with respect to their relevance in discriminating the gestures. Using a digital glove we have to extract features modelling the behavior of a group of signals. Such analysis of time series is not confined to gesture recognition but appears in very diverse fields such as medicine (heart beat features), finance,etc... Zhu [22] and LaViola Jr. present some traditional ways of generating features. These techniques largely appeal to digital signal processing (**DSP**) techniques for modelling hand temporal behaviors (Fourier transforms, wavelet transforms,...) and to statistics in order to find similarities between gesture samples (principal component analysis, singular value decomposition,...).

  After having selected relevant features, we obtain *feature vectors* corresponding to particular hand shapes. Sequences of hand shapes may be difficult to interpret for some classifier and we reduce them as an unique time series using *vector quantization*. The purpose of vector quantization is to ultimately match every given feature vector with an element of a finite set of key vectors, minimizing the distortion between the two elements of the resultant pair.

  A special attention must be brought to the representation stage since it has a direct influence on quality of the discrimination between gestures.

- All patterns to detect having been judiciously represented, we have yet to take a decision when facing a gesture input: are we in front of a pattern

and which one? To take these decisions, several classification algorithms were tried, some of which will be quickly reviewed in the coming section.

2.3.2 Gesture recognition strategies

In their incentives to capture gestures, people came up with a variety of strategies. Although LaViola reviewed very completely the domain in his work, we will briefly mention the main recognition approaches that were tried, summarized in table 2.1.

- **Fuzzy logic** is a branch of mathematics dealing with multivalued logic. Using it has been proven helpful when people wanted to represent formally notions involving human subjectivity (for instance the concept of *beauty*). Gestures, by the multiplicity of manners they can be performed (more or less quickly, with slight variations), have been considered as a fuzzy concept and expressed as logical predicates [28] . An example of fuzzy logic approach would first define some *states* for the fingers to express their configuration (straight, curved, closed). In a second time, membership functions would give the probability of a finger to be in each state. Gestures are then expressed as predicates: for instance the *pointing* sign could be approximated as the conjunction of all fingers being *closed*, with exception of the index which would be *straight*. Eventually after having evaluated the various predicates and obtained corresponding probabilities, we decide if a gesture has been recognized.

  Establishing the different predicates can be tiresome and results in a lack of flexibility in the application (new gestures imply new predicates to be programmed), therefore people coupled the fuzzy logic approach with learning paradigms such as neural networks. Once these latter are trained, they define some implicit rules for each gesture .

  Yet, fuzzy logic works fine with signs, but the larger information necessary for encoding temporal behaviors prevents it from being practical with respect to dynamic gestures.

- A natural strategy when it comes to recognize a pattern consists of constructing a model and then trying to match it with inputs. This **template matching** approach is very straightforward, usually works fine if the ges-

ture models are distant (which is more likely to occur with small gesture sets) but tends to be unpractical with dynamic gestures [24] [29].

- **Neural networks** as classifiers with a reputation of being robust to noise were well investigated for recognizing gestures. Although neural networks are usually dedicated to classification of static data (no time factor intervening), some variant can handle temporal information. Indeed gestural time series were processed using a time delay neural network by Harling [27] and Sandberg [25], recurrent architectures have been tested in [32],[33]. Both previous works are based on multilayered neural networks, however other types of networks were also researched: for example Kohonen feature maps in [35].

  Using neural networks has been successful to the extent that tens of gestures could be classified with good chances: a fingerspelling alphabet of 42 gestures with a recognition rate of 98% for Taguchi and Murakami, Fels and Hinton recognize more than 60 different hand shapes (similar recognition rate) [34].

- Finally probabilistic data structures known as **hidden Markov models** were applied with success to detect gestures [26] [37][38] [39]. This HMM paradigm is of particular interest for two reasons:

  - First, its allows to represent gestures with an architecture (*left-right*) reflecting the temporal evolution of the hand. As the hand behavior is taken in charge by the architecture, we can focus more during the representation recognition stage on extracting particularities of instantaneous hand shapes.
  - Second, by its probabilistic approach of the problem it adresses elegantly the problem of distortion: when a hand is staying longer in a particular configuration due to a slowdown in the user execution, this will be translated by a self transition of a state corresponding to the hand shape.

The previous list is not exhaustive. Listed methods envisionned gestures without their semantic meaning: the recognition may be also helped by adding some linguistic constraints (*grammar*).

Strategy	Gesture type	Reference
Recursive NN	dynamic gestures	[32]
Kohonen feature maps NN	dynamic gestures	[35]
Time delay NN	dynamic gestures	[27] [25]
Fuzzy logic and perceptron	signs	[28]
Template matching	signs	[24] [29]
HMM	dynamic gestures	[26] [37][38] [39]

Table 2.1: Some gesture recognition strategies

Possible applications

Gesture recognition is not a mind puzzle but has very practical applications: in 1989, the Nintendo^{TM} game industry proposed to use the digital *Power Glove* to enhance the interactivity [6]. Gestures were also paid attention to help physically-impaired people's communication with computers [14]. With the increase of computational power in the 90ies entailing more detailed virtual worlds, gestural inputs became an alternative of choice in certain applications, presenting some implicit communication aspects hard to verbalize: for instance in computer assisted design (**CAD**) [11], modellers are interested in gesturing for sketching shapes.

2.4 Implementation tools

2.4.1 Java3D

The Java3D Application Programming Interface (**API**) is an extension of the Java language created in 1997 by Sun [4]. One particularity of the Java3D API resides in its object oriented (**OO**) design, which makes it very convenient to use. The creation of a 3 dimensional VE relies on the specification of two graphs, which stem from a *VirtualUniverse* top node (figure 2.7):

- The **view graph**.
  This graph is represented on the right side of the figure, beginning by a *BranchGroup* node. It contains all information related to the scene viewing and rendering.
- The **scene graph**.

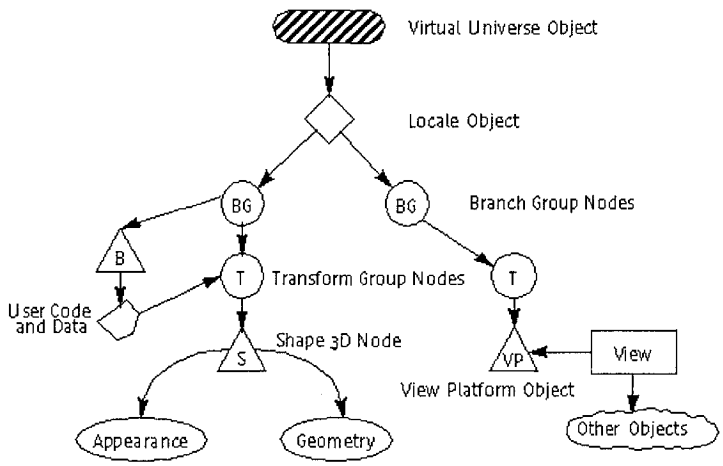

Figure 2.7: A simple Java3D architecture

This graph begins with the left *BranchGroup* on the previous figure. It specifies the virtual objects composition and spatial organization through a directed acyclic graph. The hierachy displayed by the graph is based on *parent-child* relations, signified by arrows. To prevent any circle from appearing, a node may not have more than one parent.

Among possible nodes that can be used for establishing a scene graph, we should mention:

- *BranchGroup* nodes, which are used for attaching solidary components, yet without any hierarchical dependences, together. For instance when dealing with an human avatar, the *trunc* and *arm* subgraphs would be attached to a same *BranchGroup* object.
- *TransformGroup* nodes. Inside each *TransformGroup* node is stored a *Transform3D* object, which specifies through a 4 by 4 matrix (3 by 3 rotation matrix, a 3 dimensional translation vector and a scaling value) the geometrical transformation to apply on children nodes.

  This way, Java3D takes advantage of its OO design to store the global geometric transformation of a node as a stack of elementary opera-

tions, each of them being incorporated in a parent *TransformGroup* node. Moreover this is convenient for hierarchies of objects: if an *arm* is modelled by 3 *TransformGroups* for the 3 main joints (shoulder → elbow → wrist), a change of the shoulder orientation is directly transmitted to lower components (forearm,hand,..).

- *Behavior* nodes are added for interaction purpose and perform precise actions in response of certain events (AWT Java events,etc...). The figure 2.7 is misleading in that the relation between the behavior object and its targeted object "User code and data" is represented the same way as a *parent-child* relation whereas it is only a reference relastionship (thus "T" does not have 2 parents).
- *Geometry* and *Shape3D* nodes. These nodes, that are leafs in the scene graph are used for setting the visual aspect of virtual objects. We should notice that these objects being the only ones rendered when the application is running, they are the only ones that can be *picked* (i.e. selected by a user using some Java3D procedures).

Finally, for readers interested in more Java3D implementation details, we refer them to the very complete website [5], which helped us a lot.

2.4.2 JNI

The Java Native Interface (**JNI**) is an extension of the Java language, which allows programmers to call some functionalities written in another language than Java , often c or c++ in some Java program and vice versa (figure 2.8).

The JNI was of use, since it allowed us to create a VE application using Java3D and at the same time communicate with the digital glove's driver written in c++ .

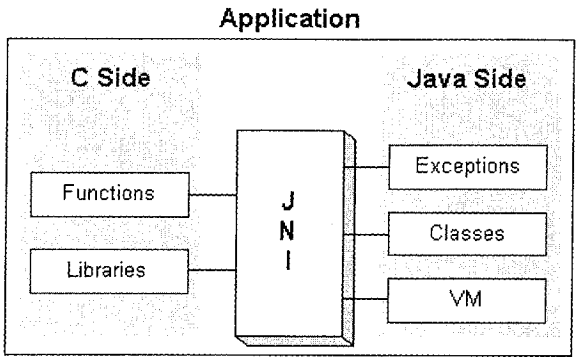

Figure 2.8: JNI role

Chapter 3

Representation

This chapter will encompass the the first stage of the GR process, namely the gesture representation step. Just after having retrieved data from the capture devices, people usually preprocess them in order to facilitate later recognition steps. As measurements have been made more exploitable, the next challenge is to decide of a gesture representational format, which will ideally allow to discern distinctly every gesture.

3.1 Retrieval parameters

Collecting data from both input devices is the very first step for our interface. Although the CyberGlove^{TM} and the tracker APIs provide us with functions to perform this task, we have yet to define a sampling rate. As we are studying hand shape gestures, the digital signal sampling theorem gives us some hint about possible sampling rates.

Sampling theorem:
To avoid aliasing, a limited bandwith signal with maximum frequency f_{max} should be sampled with a frequency f_s at least greater than twice f_{max}.
The CyberGlove ^{TM}user manual mentions that a low-pass filter with corner frequency of 30 Hz has been integrated in the analog-digital converter, allowing us to choose sampling rates below 15 Hz without facing some aliasing issue.

To see what high frequencies are involved in finger motions, we ask the user to make random gesticulations for a certain amount of time while recording the

glove data. Analyzing the spectra of the various joint signals, figure 3.1, we observe that the significant part lies below 6 Hz. Moreover the cyberglove ™ reference manual mentions a study demonstrating that open loop hand gestures' spectra are usually within 3 Hz. As a conclusion, opting for a sampling rate of 10 to 12 Hz appears appropriate.

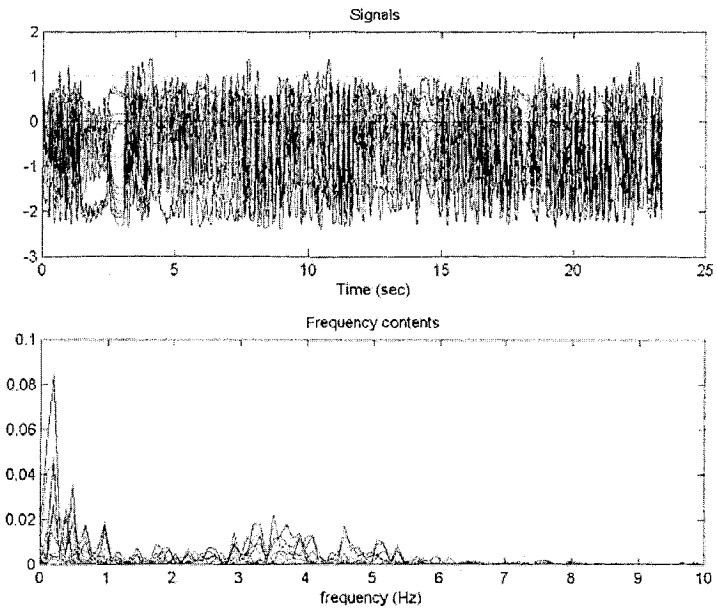

Figure 3.1: Joint signal spectra

The second retrieval parameter is the time window. We have to choose it great enough so that targeted gestures may be performed within, while not too large, otherwise more than one gesture may be executed thus hampering the detection. Our work is thus based on a **constant time window**, arbitrarily set to 1 second.

3.2 Signal normalization

As the cyberglove provides us with *raw data* reflecting the joint angle variations, we can observe different ranges among the joint signals. This is not a desirable

property, for it makes some joint *numerically* more important than others. In order to establish some numerical equity, we normalize the different signals so that their variations are between 0 and 1.

The normalization process is done in two steps. In a first time, we ask the user over a certain period of time, typically 30 seconds, to close and widen his finger joints as much as possible. The finger movements are logged then analyzed to determine the lower and upper bound of each joint signal. In a second time we normalize each joint data by using some parameterized sigmoid function. This function is defined using two parameters α and β for each signal.

$$sigmoid_{\alpha,\beta}(x) = \frac{1}{(1 - e^{-\alpha(x-\beta)})}$$

The two parameters β and α are respectively used to center the sigmoid's inflexion around the range of values where the signal varies most and although making its transition from 0 to 1 as step as the signal's variation range is narrow. We thought this approach would ensure a good sensitivity with respect to a signal's variations.

As the previous function's range is between 0 and 1, we can consider it as the probability of the coresponding joint to be *open*. Subsequently, we set two additional parameters *inf* (0.1) and *sup* (0.9) representing this probability when the signal reaches respectively its lower and higher bounds.

$$sigmoid_{\alpha,\beta}(b_l) = inf$$
$$sigmoid_{\alpha,\beta}(b_u) = sup$$

Using these parameters, we compute the α and β parameters:

$$\alpha = \frac{log(1/inf - 1) - log(1/sup - 1)}{b_u - b_l}$$

$$\beta = \frac{b_u.log(1/inf - 1) - b_l.log(1/sup - 1)}{log(1/inf - 1) - log(1/sup - 1)}$$

3.3 Correlation study

Intuitively, it is obvious that the different normalized joint signals are in some measure correlated. Indeed, simple observations of finger motions show clearly

that the four last fingers motions(*index*, *middle*, *ring*, *pinky*) are linked while the *thumb*, the only opposite finger, is relatively free. In the following lines we will perform a quantitative analysis about the finger joints interdependencies.

To measure the independence of two time series $\mathbf{x}$ and $\mathbf{y}$ of length $\mathbf{n}$, we use the normalized covariance (*ncov*) function defined with help of the covariance function (*cov*) by:

$$ncov(\mathbf{x}, \mathbf{y}) = \frac{cov(\mathbf{x}, \mathbf{y})}{\sigma_{\mathbf{x}}\sigma_{\mathbf{y}}} \tag{3.1}$$

$$cov(\mathbf{x}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \mu_{\mathbf{x}})(\mathbf{y}_i - \mu_{\mathbf{y}}) \tag{3.2}$$

As the covariance computations must be done on a time series varying as much as possible so that the influences among them could be unveiled, we used the signals from the *normalization log* requested from the user. The results of these computations are stored in the table 3.3. The 20 joints captured with the cyberglove^{TM} are referred with the alphabetical letters correponding to their index in the array obtained from the cyberglove ^{TM}. If for each finger we list the joints starting from its base to its tip, we obtain the following translations:

- *Thumb*: a, b, c
- *Thumb-index* : d
- *Index*: e, f,g
- *Index-Middle*: h
- *Middle*: i, j, k
- *Middle-Ring*: l
- *Ring*: m, n, o
- *Ring-Pinky*: p
- *Pinky*: q, r, s

The table values were rounded with a precision of 1% and we highlighted the ones greater than 90 percent. For clarity sake, as the resulting covariance is symmetric, we only deal with the upper diagonal part. As expected from the empirical finger motion study, we observe some very high rates among the 4 palm fingers compared with the thumb ones:

- The thumb three joints' normalized covariance rates are at most of 60%.
- All other finger or abduction joints can be grouped into pairs with rates between 94% and 100%. Excluding the thumb joints a,b and c, we sorted the correlation rates in descreasing order and matched every single joint signal with another when their dependence was of at least 90%. Eventually every joint signal was matched with at least a normalized covariance rate of 94% in absolute value:
  - [d,p] [g,q] [j,r] [m,s] (100%)
  - [d,h] [i,l] (98%)
  - [e,i] (97%)
  - [f,o] (96%)
  - [k,n] (94%)

  The high dependencies between the aforementionned joints can be visually acknowledged by looking at figure 3.3. In this picture, in spite of some outliners, we can distinctly discern some more or less thick segments indicating high linear correspondencies between two signals. Looking at some other signal pairs, we may find some visually satisfying dependencies such as in figure 3.2, however their normalized covariance rates are usually not high (60% maximum). Indeed, if we look at the points figuring q-r signals' measurements, beside a short segment we can observe many *outliners*.

Although, this study demonstrated some clear dependence between the joint signals, it has not been used to reduce the number of signals to analyze in order to capture gestures.

3.4 Feature extraction

This section will take on the preprocessing previously done and strive to represent gesture entities as clearly as possible. A *clear* representation in our recognition context is one that would allow to clearly differentiate gestures from noisy hand shapes (detection) and to discriminate easily the gestures (classification).

Although the precedence of detection over classification is from a theoretical point of view desirable, it is hard to put in place because of the segmentation

Joints	a	b	c	d	e	f	g	h	i	j
a	**1**	0.59	0.04	-0.43	-0.25	-0.4	0.33	-0.42	-0.35	0.6
b	0.59	**1**	0.72	-0.1	0.12	-0.08	-0.11	-0.12	0	0.54
c	0.04	0.72	**1**	0.1	0.31	0.2	-0.34	0.07	0.23	0.3
d	-0.43	-0.1	0.1	**1**	0.48	0.37	-0.57	**0.98**	0.5	0
e	-0.25	0.12	0.31	0.48	**1**	0.89	-0.47	0.52	**0.97**	0.01
f	-0.4	-0.08	0.2	0.37	0.89	**1**	-0.44	0.41	0.89	-0.2
g	0.33	-0.11	-0.34	-0.57	-0.47	-0.44	**1**	-0.55	-0.44	0.18
h	-0.42	-0.12	0.07	0.98	0.52	0.41	-0.55	**1**	0.53	0.02
i	-0.35	0	0.23	0.5	0.97	0.89	-0.44	0.53	**1**	-0.13
j	0.6	0.54	0.3	0	0.01	-0.2	0.18	0.02	-0.13	**1**
k	-0.25	-0.08	-0.02	0.91	0.32	0.17	-0.36	0.93	0.3	0.23
l	-0.43	-0.09	0.18	0.52	0.94	0.92	-0.46	0.55	0.98	-0.21
m	-0.51	-0.09	0.21	0.52	0.43	0.48	-0.95	0.5	0.44	-0.49
n	-0.2	-0.11	-0.09	0.79	0.23	0.05	-0.11	0.82	0.22	0.3
o	-0.41	-0.1	0.17	0.49	0.91	0.96	-0.51	0.52	0.92	-0.21
p	-0.43	-0.1	0.1	1	0.48	0.37	-0.57	0.98	0.5	0
q	0.33	-0.11	-0.34	-0.57	-0.47	-0.44	1	-0.55	-0.44	0.18
r	0.6	0.54	0.3	0	0.01	-0.2	0.18	0.02	-0.13	1
s	-0.51	-0.09	0.21	0.52	0.43	0.48	-0.95	0.5	0.44	-0.49

Joints	k	l	m	n	o	p	q	r	s
a	-0.25	-0.43	-0.51	-0.2	-0.41	-0.43	0.33	0.6	-0.51
b	-0.08	-0.09	-0.09	-0.11	-0.1	-0.1	-0.11	0.54	-0.09
c	-0.02	0.18	0.21	-0.09	0.17	0.1	-0.34	0.3	0.21
d	**0.91**	0.52	0.52	0.79	0.49	**1**	-0.57	0	0.52
e	0.32	**0.94**	0.43	0.23	**0.91**	0.48	-0.47	0.01	0.43
f	0.17	**0.92**	0.48	0.05	**0.96**	0.37	-0.44	-0.2	0.48
g	-0.36	-0.46	**-0.95**	-0.11	-0.51	-0.57	**1**	0.18	**-0.95**
h	**0.93**	0.55	0.5	0.82	0.52	**0.98**	-0.55	0.02	0.5
i	0.3	**0.98**	0.44	0.22	**0.92**	0.5	-0.44	-0.13	0.44
j	0.23	-0.21	-0.49	0.3	-0.21	0	0.18	**1**	-0.49
k	**1**	0.3	0.26	**0.94**	0.27	**0.91**	-0.36	0.23	0.26
l	0.3	**1**	0.49	0.21	**0.96**	0.52	-0.46	-0.21	0.49
m	0.26	0.49	**1**	0.01	0.54	0.52	**-0.95**	-0.49	**1**
n	0.94	0.21	0.01	**1**	0.14	0.79	-0.11	0.3	0.01
o	0.27	0.96	0.54	0.14	**1**	0.49	-0.51	-0.21	0.54
p	0.91	0.52	0.52	0.79	0.49	**1**	-0.57	0	0.52
q	-0.36	-0.46	-0.95	-0.11	-0.51	-0.57	**1**	0.18	**-0.95**
r	0.23	-0.21	-0.49	0.3	-0.21	0	0.18	**1**	-0.49
s	0.26	0.49	1	0.01	0.54	0.52	-0.95	-0.49	**1**

Table 3.1: Intersignal covariance rates

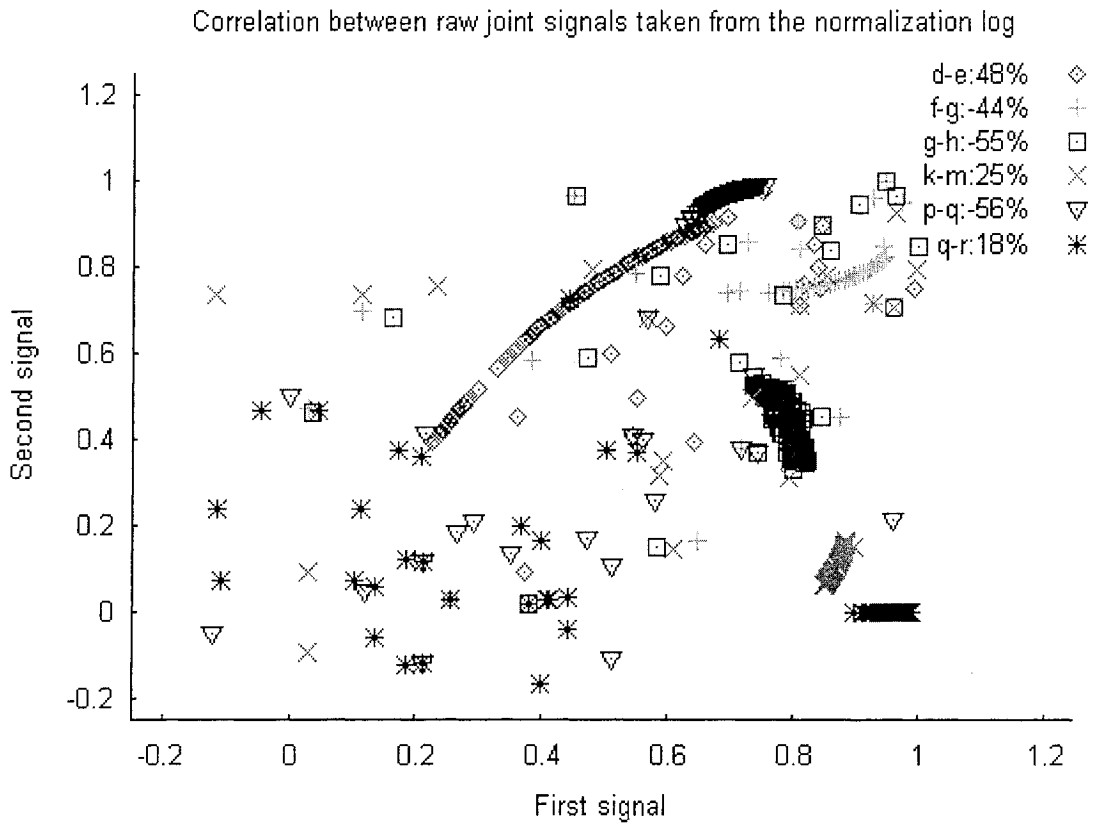

Figure 3.2: Some visual signal dependences

problem. Practically, we prefer to classify every input, then assert the probability of the recognized gesture. If the probability may be considered high enough (see Classification chapter), we deem the corresponding gesture as having been performed.

A suitable gesture description relies on highlighting commonalities shared by various samples of one gesture and then on retaining only those that differ significantly among gestures. The gesture characteristics that we consider in these two steps are refered in the literature as *features*. A feature according Miao [16] is any extractable piece of information numeric or symbolic. When capturing gestures through a digital glove, the joint angles measurements are examples

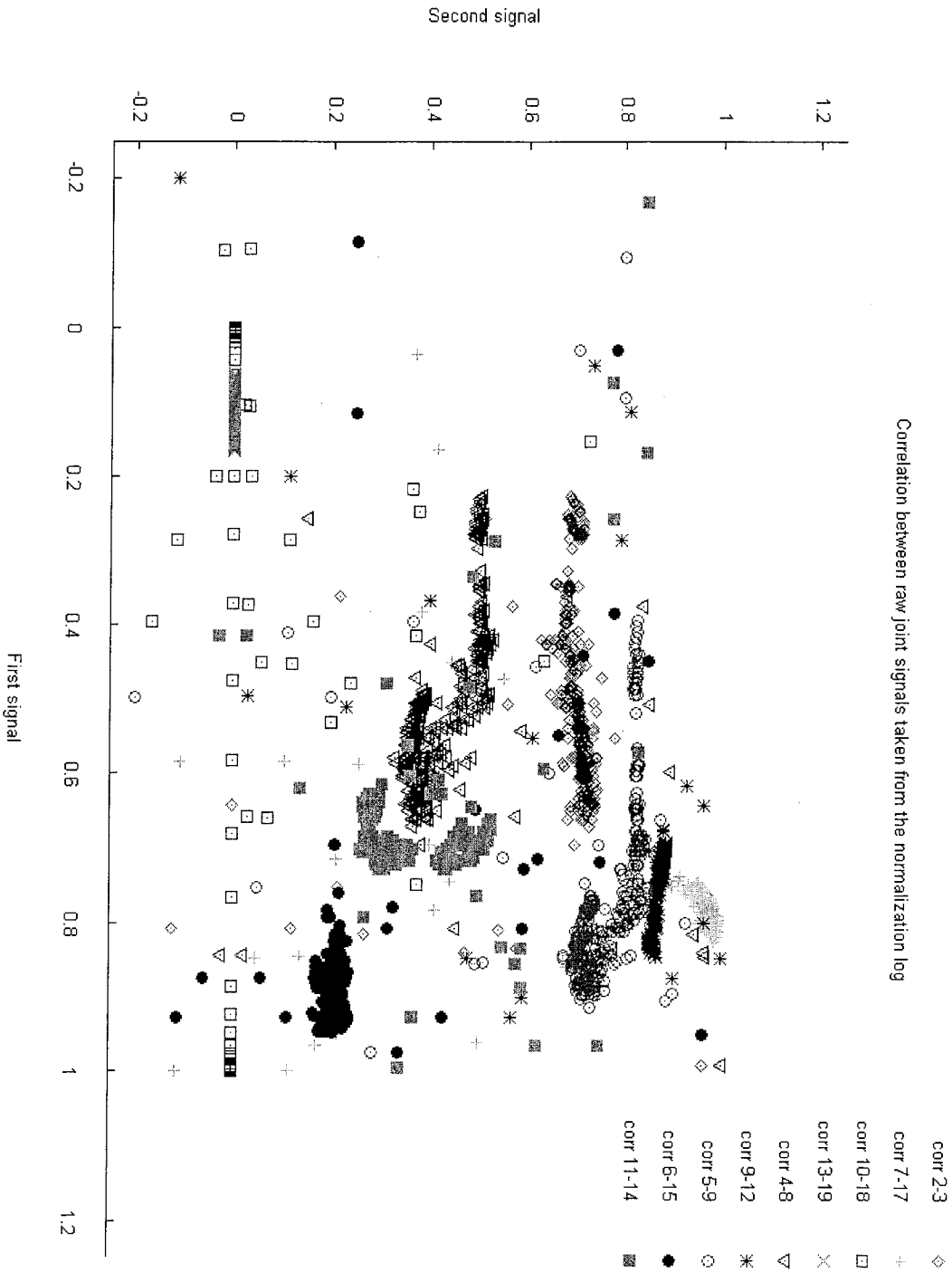

Figure 3.3: High covariance among some joint signals

of numeric features, whereas the finger states such as *closed,bent,straight* are symbolic informations. If the gestures were captured by a video camera, neither sort of features would be directly available.

In our incentive to characterize gestures, we came up with two different sorts of features. The first kind, namely *static* features, would describe instantaneous hand shapes, while the second sort,*temporal* features, would give informations about the signals evolutions. Static features are sometimes referred in the literature as *spatial* features for they give spatial information about the hand configuration and no temporal information. We chose not to retain this expression, to avoid possible ambiguities with the hand spatial motions.

To assess the quality of a feature, we should look at the discrimination between gestures and also the intra-class dispersions it provides. A feature's descriptive value is good when the former (inter-gesture discrimination) is maximized and the latter is low for every gesture class.

Feature generation

As we previously said, the normalized joint measurements can be considered as numerical features. By default, these values of equal importance (normalization) will compose our feature vector. When using more imagination to extract new features from the joint measurements, we should make sure that the resultant values are also normalized otherwise it could bias the classification process by making some features numerically preponderant.

- **Static features**:

  The objective of static features is to characterize hand shapes at one point in time. Presently, the normalized joint measurements show information about joints openesses. Yet, in daily life when asked to describe a sign we do not give the detail of each joint configuration but use some high-level description such as *stretched*, *crossing* fingers [28].

  There certainly are numerous manners to qualify a hand's current state but we limited ourself to finger's states. Adopting a similar attitude as with joint normalized values, we will affect a value to each physical finger that should range from 0 to 1. If the value is null, then all joints of the finger would be close and conversely when the value will equal 1, the finger will be totally stretched.

When considering 3 finger joints, a straightforward way of obtaining such value would be to multiply their measurements. However, from the brut multiplication, we have no clue about the actual finger configuration: by exchanging the joint state values, it would yield the same result but may correspond to a dramatically different finger configuration.

The previous observation oriented us to look for a function f whose results would give a definite idea about the finger configuration. Let x_i, i=1..3 be the three finger joint float measurements (base, middle, tip) and s be a discretization step delimiting different joint states.

$$\begin{aligned} N &= [\frac{1}{s}] \\ \forall i \in [1..3], n_i &= [\frac{x_i}{s}] \\ f(x_1, x_2, x_3) &= \frac{1}{(N+1)^3 - 1}((N+1)^2 * n_1 + (N+1) * n_2 + n_3) \end{aligned}$$

The function f defined above with the help of the floor function ([x] = floor(x)) returns a value between 0 and 1, that theoretically reaches 1 when all joints are wide open. As this function definition does not treat every joint equally (the base joint is preponderant with a coefficient of $(N+1)^2$), given a result value we can backtrack to the n_i values through the use of congruence operations. We did not treat each joint equally for some perceptual reasons: for a same angle, the closer to the finger base a joint is the more influence on the finger state (*close / open*) the joint would have.

To decide on a discretization step s, we looked at the importances of the joint states when computing f (table 3.2).

We judged that the values of 2, 3 ($s = 0.5$ and 0.6) corresponded best to the influence of each joint's in the visual perception of a finger's state.

- **Temporal features**:
  Our hand shapes temporal variations are reflected by various joint signals. Since the joint signal synchronization aspect is taken care by static

N values	2	3	4
Base joint	70 %	76 %	81 %
Middle joint	22 %	19 %	16 %
Tip joint	8 %	5 %	3 %

Table 3.2: Relative importance in percents of the finger joints

features, we need to describe each signal individually. The digital signal processing branch of mathematics offers a vast range of solutions for this problem: Fourier analysis, wavelets decompositions,etc ... Because of some implementation considerations, we did not explore them and favored instead basic signals aspects such as:

- One simple and coarse manner to give some information about a signal's outlook is to compute its **mean** over a certain period of time. We should notice that by averaging normalized values, we also obtain results between 0 and 1. The idea of using some temporal window averages, came to eliminate small fluctuations of joint measurements, which are unecessary for signs.
- Once we know the mean value of a signal over some time window, we would like to see its fluctuations. Dynamic gestures, especially the ones that are cyclic such as the *scissors* gesture, are likely to have high variances for some joint signals; significant variances could be used as a signature for differentiating dynamic gestures and postures.

  Additionally, we can notice that the normalization condition is also satisfied when computing the standart deviations of the signals.

Previously, we insisted on how important it was dealing with normalized temporal signals. Although every feature that we extract has its value between 0 and 1, the feature signals variation ranges may differ: for instance a finger mean signal could range from 0.3 to 0.9, whereas variance signals ranges would be restricted between 0 and 0.003. When these range differences are great, gesture representations are predominantly based on the feature with the widest range (*mean* in the previous example). Thus, we decided to make each feature varying between 0 and 1 included.

Feature selection

As we came up with a very limited number of features, we did not think necessary to engage a *feature selection* procedure. As their number increases, such procedure might be appropriate: it would reduce the redundancy among features, and by limiting their number, it may help the recognition process' performance. Techniques for selecting features based on their discriminative powers and correlations are described in [17].

3.5 Vector quantization

Once the feature extraction has been decided, each measurement retrieval that we process results in a feature vector of generally important dimension. The temporal behavior of the hand is distributed along the different components of the feature vectors. This distribution complicates the necessary analysis for unveiling temporal behaviors similar to our gestures. One way of reducing the analysis complexity is to apply some vector quantization technique (**VQ**): examining the different feature vectors occuring in our various gesture samples, we construct a set of feature vectors, which will approximate all others. This set is called a *codebook*. The different feature vectors composing the codebook are referred as *codewords*. Using the codebook, we encode our gesture samples as sequences of codewords (each codeword having been labelled with a distinct number). We have eventually represented each gesture sample as an unidimensional signal. For instance, an encoding such as $\{1, 1, 1, 2, 2, 6, 6, 6\}$ should be understood the following way: the 3 first feature vectors extracted at the 3 first time steps of the sample are approximated by the first codeword whereas the two feature vectors in line are expressed by the second codeword and the sixth codeword provides the best approximation for the feature vectors of the remaining time steps.

3.5.1 Codebook

The principle of vector quantization is simple. Yet building a *good* codebook is hard. Ideally samples of a sign should be encoded as sequences of the same codeword (the hand is poised), whereas samples of a dynamic gesture should be encoded with more than one codeword (hand shape variation), the codewords

being recurrent between the samples:

Sample 1: { 0,0,0,1,1,2,2,2,2 }
Sample 2: { 0,0,1,1,1,1,2,2,2 }

- **Brut clustering strategy:**

  Possible codebook sizes range between 1 and the number of different feature vectors generated by our gesture samples (*nbFeatureVectors*). The ratio of the codebook size over *nbFeatureVectors*) will be termed as *reduction rate*. A reduction rate neighboring zero percent will mean that most feature vectors are represented by their own codeword. Although different gestures will certainly be encoded differently, it is to fear that commonalities between their samples would be ignored. On the contrary, if the reduction rate is close to 100 %, all feature vectors will be approximated by a small number of codewords, restraining the freedom to encode gestures, therefore hampering the discrimination between gestures. In the worst case, all samples would be encoded by a sequence of an unique codeword.

  Consequently, choosing a reduction rate is equivalent to balancing the discrimination between gestures and the similarities among samples of each gesture. Once the choice has been made, we use the *k-mean* clustering algorithm to define the codewords. First, we seed as many codewords as implied by the reduction rate with different feature vectors. Then, we apply the k-mean algorithm iterative procedure on the whole set of feature vectors. This latter set contains all feature vectors present in our samples, possibly with multiplicities greater than one. The maximum variation tolerated for the centroids, that will form our codewords, is set to a fraction of the maximum variation observed between two consecutive feature vectors in samples.

  An obvious advantage of this brut clustering approach to elaborate the codebook is the concision it presents. Indeed, as samples are not processed individually but collectively,

- **Recursive approach**

Another possibility to obtain a codebook consists of using the *Linde - Buzo - Gray* (**LBG**) algorithm, explained further. This time, all centroids stem out an original codeword, set as the barycenter of our feature vector set, which is recursively divided by two until the codebook size is reached. The *top-down* approach adopted by this algorithm constrasts with the previous algorithm, whose codewords were seeded among the available feature vectors.

3.5.2 The K-mean algorithm:

Data clustering is in itself a vaste area for which many algorithms were developed (see [17]). For our needs, we picked up the popular *k-mean* algorithm, which will be described in the next lines. The essence of the algorithm is to position iteratively **n** vectors in a vector set, so that they could be viewed as barycenter of some vector groups. This algorithm relies on a metric, which we chose to be euclidian for all features are considered of equal importance.

The steps involved in the algorithm are:

1. **Seeding**:
   This step is a preliminary one, in which the user has to hand pick **n** arbitrary vectors, which will be later hold as the cluster centroids. Naturally, all feature vectors are not suitable for initializing the centroids: for instance, if we pick **n** close vectors located *far* from our vector set, we might see them converging to a unique group of vectors. When problem knowledge (vector set shape) is not available, an acceptable way to proceed would be to choose **n** vectors from the set (ideally remote from each other).

2. **Cluster update**:
   Examining the vector set, we determine for each element its closest centroid. Once the nearest centroid is found, the vector is considered as belonging to the centroid's cluster.

3. **Centroid update**:
   Previously, we updated all **n** clusters; as centroids must be the barycenter of their clusters, there may be some inconsistencies. Going through each cluster, we set their centroid as the barycenter of the elements taking

the multiplicities of the cluster elements into account. Moreover, we keep track of the distance between the former locations of the centroids and their newest ones.

The algorithm running procedure starts with the first step. Then we perform successively the last two steps until the centroids variations are judged small enough. Then the clusters are deemed as stable.

3.5.3 The Linde - Buzo - Gray algorithm

This algorithm has been presented in 1980 (described in [36] [9]) and is specifically designed for vector quantization. The main idea is to approximate in an initial phase all vectors by their barycenter, which we establish as the first codeword. This algorithm relies on the idea of quadratic *distortion*, which is the average square distance (euclidian in our case) of the vectors and their approximations. Then as long as the distortion has not been sufficiently reduced, we split each codeword in two (the two resulting codewords remaining in a close neighborhood of their parent), and iteratively position the new codewords as barycenters of the closest vectors. Experimentally, we set *epsilon*, an arbitrary small value used by the algorithm, as a fraction of the maximum distance between two successive feature vectors in our gesture set.

3.5.4 Feature vector classification

Once the codebook has been constructed, we can partition the feature vector space $\mathcal{F}$ into different Voronoï cells V_i, each of them corresponding to a particular codeword. If $\mathbf{n}$ is the size of our codebook, $\mathbf{d}$ the euclidian metric, the Voronoï cell $\mathbf{V}_i$ for the codeword $\mathbf{w}_i$ is defined as follows :

$$V_i = \{\mathbf{x} \in \mathcal{F} |\ d(\mathbf{x}, \mathbf{y_i}) < d(\mathbf{x}, \mathbf{y_j}),\ \forall j \neq i\} \tag{3.3}$$

However, a completely partitionning $\mathcal{F}$ is not what we truly desire, since we would like to consider arbitrary hand shapes, not figuring in our gesture samples, as noise. Considering the figure 3.4, we observe that some Voronoï cells such as **A** may be bounded whereas the cells on the *outside* are limitless. Feature vectors that do not translate any meaningful hand configuration are in unbounded cells, far from the codewords. When given an arbitrary feature vector $\mathbf{f}$, we proceed in two steps:

1. We find $\mathbf{w}_{closest}$, the closest codeword. The vector $\mathbf{f}$ is therefore in $\mathbf{V}_{closest}$.

2. Let $d_{furthest}$ be the half of the greatest distance of $\mathbf{w}_{closest}$ with all other codewords. If the $d(\mathbf{f},\mathbf{w}_{closest})$ is greater than $d_{furthest}$, we dismiss $\mathbf{f}$ as noise, otherwise hold it for a meaningful hand configuration.

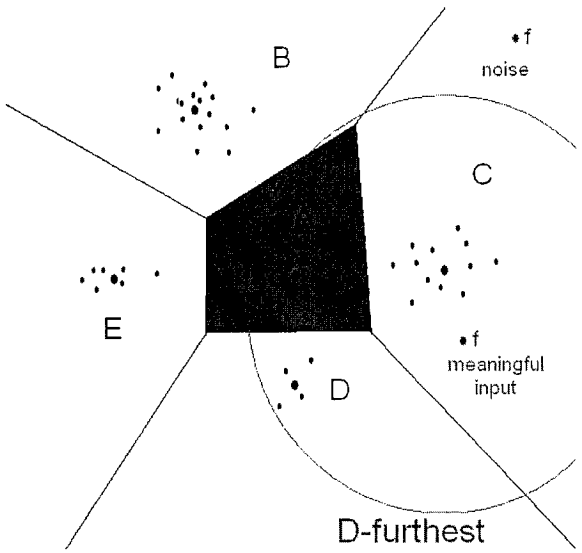

Figure 3.4: A 2D feature space, partionned into several Voronoï cells

3.5.5 Encoding considerations

One important objective of the vector quantization process is to obtain eventually coherent gesture encodings. We deem encodings to be consistent for a gesture when all its samples exhibit some similar traits. The fuzzy notion of trait is hard to define:

- For **signs** we expect every sample to be encoded as a repetition of the same codeword. Ideally, it a unique sample should be used for all samples but this is not necessarily the case since the user can slightly change its hand configuration between samples. However, there shoul not be too

many variants for the same posture. Moreover, we should not tolerate that a codeword encode the samples of two distinct signs.

- For **dynamic gestures**, every sample is expected to present more than one codeword, otherwise they could be mistaken as postures. Similarly to signs, we would like to see the same codewords appearing in the different samples, possibly in the same order.

An observer may notice that the desirable aspects of signs and dynamic gestures follow forces in opposition. The more concise the codebook we have, the more uniform will be the sign encodings with the danger of using the same codeword for different signs. Ultimately the concision can be pushed as having the codebook containing a unique codeword: all feature vectors belonging to its cluster. That case shows clearly how concision may hamper the representation of dynamics in some dynamic gesture encodings since these latter would display the same repetition of the codeword. Conversely, if the codebook building phase does not reduce the number of feature vectors, the slightest change occuring in the hand configuration when performing a dynamic gesture will be represented. However, natural hand tremors would also be displayed in encodings of signs. Too much detail obfuscates the clarity of gesture representation and therefore is not attractive.

Between these two extremes, we satisfy ourselves with a trade-off that would offer enough concision to distinguish at first glance the different gestures and also propose enough details for the dynamics of some gesture. This trade-off can be symbolized by a percentage $P_{codebook}$ expressing the codebook concision. Such percentage comes from the ratio of the codebook size over the number of different feature vectors present in our gesture set.

When estimating that encodings of both signs and dynamic gestures are acceptable for a given percentage $P_{codebook}$, we engage in some refinement procedure. The raw encodings were obtained by operating the k-mean algorithm, which clusters the different feature vectors. At that stage we should notice that this algorithm, because of its seeding phase which is randomized, is not deterministic. Thus, we may obtain different gesture encodings from codebooks built for the same reduction percentage.

Chapter 4

Classification

In the previous chapter, we tried to capture the essential features of gestures in order to help us decide, observing the input streams, whether the last sequence of hand configuration corresponds to a meaningful gesticulation. In the present chapter, we will successively examine 3 types of classifiers: a template matching approach, a neural network and a hidden Markov model.

4.1 Template matching

4.1.1 Introduction

The most natural approach for classifying an object in some object class is based on comparison. First, the object is compared to each class and a degree of similarity is deducted. The comparison can be done by looking how alike are the object and each class members. For classes with a great number of objects, this one-by-one comparison might be time consuming. An alternative way would require condensing all the class information in one *template*, i.e. model, and examining how much do the template and the object have in common. Eventually, weighing the different similarities of the object to each class, we decide of its belonging.

The simplicity of this classification scheme, that we will apply for gesture objects, nevertheless requires some precisions:

- Indeed how should we build a gesture class model?
  This issue of formatting inputs adequately, so that characteristic traits

of gestures are highlighted has been introduced in the previous chapter. Once all samples of the same gesture are formatted, we can observe that some differences still remain yet their overall aspects look similar. Figure 4.1 displays 3 samples partially represented by their 10 first joint signals (sampling rate = 12 Hz, time window = 1 second) of the *grabbing* gesture. The initial and final states of the joints look very much alike yet the transition usually do not occur at the same time during the recording.

Setting one sample as *the* model would be an arbitrary decision. Instead we decided to construct a gesture template by averaging all samples. Figure 4.2 displays the average of the three partial samples from above.

The template we obtained kept the general outline of the 3 samples and we can observe that averaging tends to produce a smoothing effect. Smoothing may be troublesome when dealing with a gesture with quick variations (for instance in the *scissors* gesture): if samples' signals are in opposition phase at some time, averaging may annul their variations, therefore making us lose a characteristic of the gesture.

- What similarity measure do we use?
  Choosing a metric d or a similarity measure s may be considered equivalent, since we can relate both using an equality such as $s = \frac{1}{1+d}$. For our experiments, we investigated the following metrics:

  - *Euclidian metric:*
    Given two vectors $\mathbf{u}$ and $\mathbf{v}$ of same length n, their euclidian distance is defined by:

    $$d_{euclidian}(\mathbf{u},\mathbf{v})^2 = \sum_{k=1}^{n}(\mathbf{u}_k - \mathbf{v}_k)^2$$

  - *Mahalanobis metric:*
    In the euclidian metric above, all vector elements have the same importance. In some cases, we would like certain directions of the vector space to have more influence than others. This can be performed by affecting strictly positive weights to directions:

    $$d_{weighted\ euclidian}(\mathbf{u},\mathbf{v})^2 = \sum_{k=1}^{n} w_k\ (\mathbf{u}_k - \mathbf{v}_k)^2$$

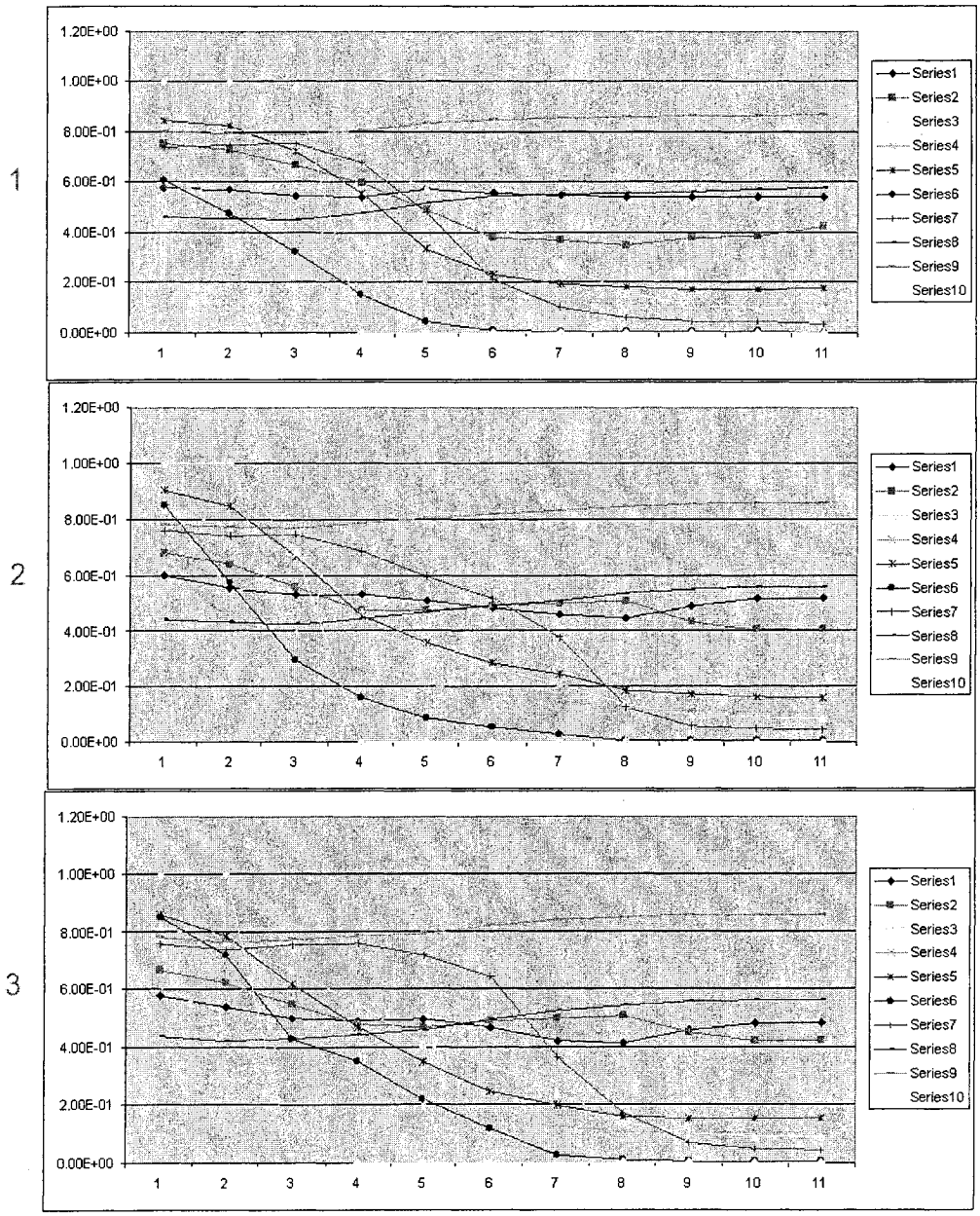

Figure 4.1: Three partial *grabbing* samples

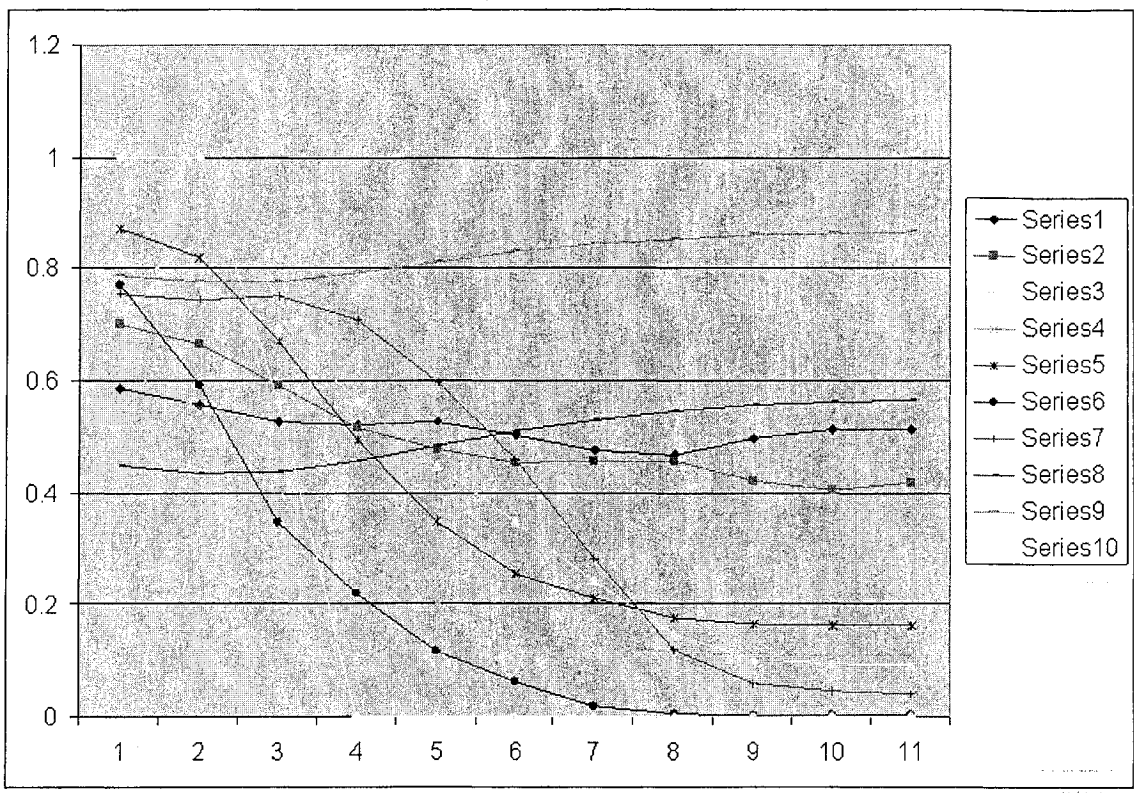

Figure 4.2: The resultant partial average

The Mahalanobis metric, that we will introduce, is a particular type of weighted euclidian distance where weights are decided with respect to some statistical considerations. The idea supporting this metric is that the less scattered samples of a class are along one direction, the more representative is the direction and thus the more weight it should have in this weighted euclidian metric. Practically, we have to compute the covariance matrix of the sample vectors, then determine its principal axis by diagonalizing the matrix and affect weights inversely proportional to the diagonal coefficients.

As weights are only based on statistical information pertaining of a unique gesture class, this metric is only meaningful with respect to the corresponding class centroid. Consequently, once given a vector $\mathbf{u}$ in the gesture sample space, we have to compute as many metrics $d^i_{Mahalanobis}$ as gesture classes we have. Then, for each class i, with covariance matrix Σ_i , we compute the distance between $\mathbf{u}$ and the class 'centroid μ_i using the appropriate metric.

$$d^i_{Mahalanobis}(\mathbf{u}, \mu_i)^2 = (\mathbf{u} - \mu_i)^t \ \Sigma_i^{-1} \ (\mathbf{u} - \mu_i)$$

However, from the previous definition arises the problem of the covariance matrix Σ invertibility. Assuming that the feature space is of dimension d, that our gesture classes are defined by n samples where $n \ll d$, this leaves us no chance of having the covariance matrix invertible (its rank would be at most n). To adress this issue, we operated a slight modification, respecting the spirit of the Mahalanobis metric: in the basis $\mathcal{B}$, where the covariance matrix can be expressed as a diagonal one, we replace the null diagonal coefficients by an arbitrary small value ϵ. Let D' be the new diagonal matrix and O the basis changement matrix; our new expression for the Mahalanobis metric is then

$$d_{approxMahalanobis}(\mathbf{u}, \mu)^2 = (\mathbf{u} - \mu)^t \ O^t \ (D')^{-1} \ O \ (\mathbf{u} - \mu)$$

One obvious limitation of the Mahalanobis metric comes from the statistical information used to compute the coefficients of this weighted euclidian metric, which limits its use to only compute the distance between a vector and a class' centroid.

– *Dynamic time warping* (**DTW**):
This paragraph will introduce a dynamic programming algorithm that will help us to capture the similarity between two signals. If we observe signals corresponding to a gesture performed in an *identical* manner at different times, we might observe some slight differences between them, that are the result of little variations in the execution speed. In a general case, changes in execution speeds cause the resulting signals to have different lengths (assuming that the sampling rate is the same). Therefore, the traditional euclidian metric would be inapplicable to uncover the similarity between them. Similar characteristics between signals may be uncovered if we give up the regular one-to-one euclidian matching, but instead warp the signals locally (see figure 4.3) respecting the following rules:

  * **Coincidence**: The starting and ending states of each signal correspond.
  * **Correspondence**: Every state of both signals has to be matched. If this condition is not respected, some major difference between both signals may be overlooked (case **a** in figure 4.4).
  * **Temporal irreversibility**: Once a state has its successor matched, it cannot be reused for a later matching. If this hypothesis is ignored, a signal's part can be matched twice or more, therefore missing some dissimilarity (case **b** in figure 4.4).

When we respect these properties, signals that present same sequences of values (in the example: first 1 then 2, then 3 and last 4) but may spend different times in each value are considered as identical:

$$\begin{aligned} s_1 &= \{1,1,1,2,3,3,3,4,4\} \\ s_2 &= \{1,1,2,2,3,3,3,3,4,4,4\} \end{aligned}$$

Mathematically, if $\mathbf{u}$ and $\mathbf{v}$ are two vectors representing signals. n_u,n_v their respective lengths, the DTW algorithm unfolds as fol-

lows:

$$\begin{cases} d^2(i,j) & = & (\mathbf{u}_i - \mathbf{v}_j)^2 \\ \gamma(0,0) & = & d^2(0_0) \\ \gamma(i,j) & = & min\{\gamma(i-1,j), \gamma(i,j-1), \gamma(i-1,j-1)\} + d^2(i,j) \\ d_{dtw}(\mathbf{u},\mathbf{v}) & = & \gamma(n_u, n_v) \end{cases}$$

Although the dtw algorithm is no metric, for the *separability* condition is obviously not verified ($d_{dtw}(\mathbf{u},\mathbf{v})=0$ does not imply that $\mathbf{u}=\mathbf{v}$), we will abusively refer to it as the dtw distance.

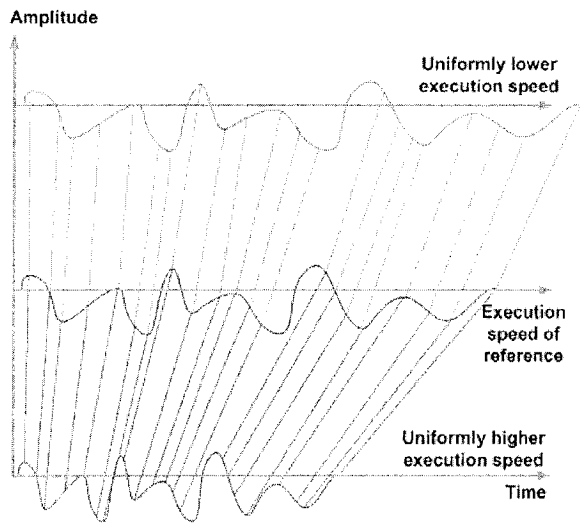

Figure 4.3: Irregular matching detecting similarities between *identical* signals performed at different speeds

As we have recalled briefly the main characteristics of the DTW distance, we must take into consideration the context, in which it will be applied. Indeed, as it is used in the template matching classifier, its purpose is to decide what gesture may possibly be represented by a sequence of feature vectors (figure 4.5).

When getting such a sequence of feature vectors, a first idea would be to concatenate all the vectors and apply the dtw algorithm. However, this is not reasonable, since the DTW algorithm might match elements of

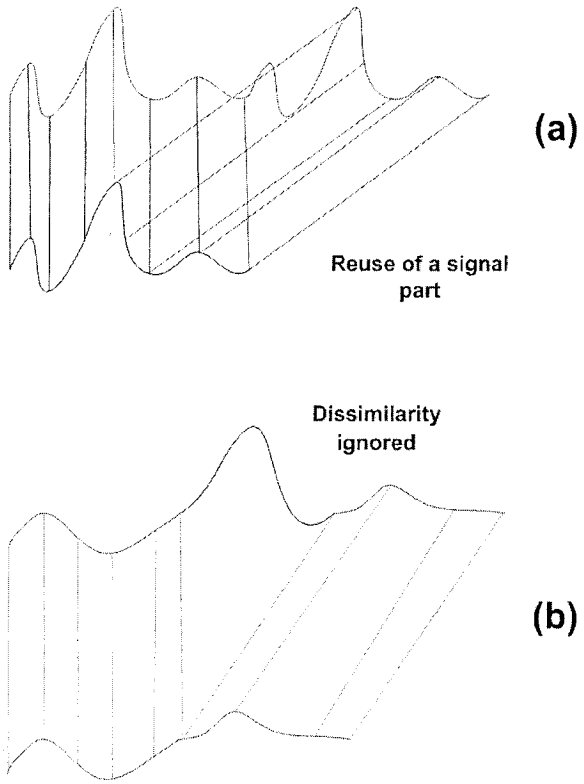

Figure 4.4: DTW constraints: (a) correspondence; (b) temporal irreversibility

different feature signals in order to minimize the temporal distortion. A more judicious approach would require first extracting the different feature signals and then applying the DTW for each of them.

If we decide to quantize feature vectors, we only need to apply the DTW once on the sequence of codeword indices. In that case, since we are not interested in distances between indices of codewords but between them, we have to modify slightly the algorithm with :

$$d^2(i,j) = \begin{cases} penalty & \text{if i or j equals -1} \\ d^2_{euclidian}(codebook[i], codebook[j]) & \text{otherwise} \end{cases}$$

The *penalty* for noisy inputs was set to twice the maximum square euclidian distance between two elements of the codebook.

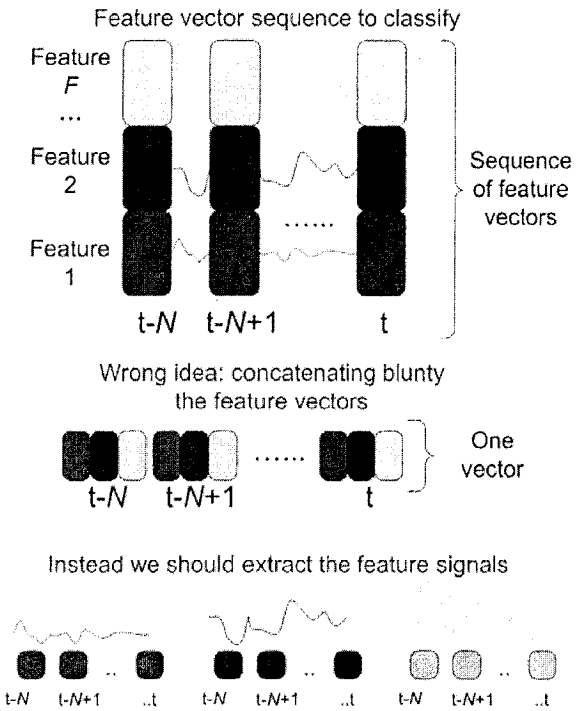

Figure 4.5: DTW with feature vectors

- How much similarity should there be to state that a gesture has been identified?

  In order to decide of **similarity thresholds** for each gesture class, we must find a tradeoff between detection sensitivity and false acceptance. On the one hand, a too low threshold will reduce gesture detection possibilities since it would restrict them to configurations very close to centroids. On the other hand, too high thresholds will allow inadequate configurations to be detected (false acceptance or misclassification).

  As a gesture's samples were captured with special care, we would like to see all of them falling in the gesture's class. In order to obtain such result, we can set the thresholds to the maximum distance between the samples and the class' model. However, with this *max* threshold strategy, we are likely to include possible outliners and create some missclassification problem:

as the gesture classes are more vaste, overlapping zones between them are more probable. We also chose two other strategies for the thresholds to compare their effects with the previous one: in the first one, similarity thresholds are set to the *mean* distance of samples and the class model while in the second they are set as the mean plus the *variance*.

Considering this three strategies in the *mean*,*variance* and *max* order , the classes' radii they generated are in increasing order. Thus, with the *mean* (respectively *max*) choice, we are likely to run in some non detection (resp. missclassification) problems.

4.2 Neural network

4.2.1 Introduction

In the 1950ies, artificial intelligence researchers have looked at Nature for inspiration. They were particularly interested by the functionning of cells located at the core of the cognition process: the *neurons*. Neurons are highly interconnected nervous cells: electric signals are transmitted via synapses to the dentrites, which propagate them to the center of the cell. If the incoming electric stimulations reach a certain level, then the cell fires an electric impulse wich is carried to other cells by the axon.

Several models of artificial neural networks were elaborated from these observations, among which *perceptrons* that we used for our gesture classification problem. This type of artificial neural network has a layered architecture, each layer containing some computational units, i.e *neurons*. Neurons of different layers may be connected by links to which weights w are affected. A neuron basic operation is defined by a *logistic* function f, that is used to compute its output value. The argument of this function is either the sum of the weighted output value of neurons connected to the input, or values set arbitrarily when this latter is not connected to any neurons.

Logistic functions are often squashing functions, so that the output value is between 0 and 1. A simple example of such function would be the *sign* function returning 0 if its argument is negative, 1 otherwise. However, due to its discontinuity in 0, smoother functions such as the *sigmoid*, *hyperbolic tangent*

(tanh) are preferred:

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$
$$tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$$

Frequently a bias is added to the sum of weighted output to compute the argument of the logistic function. This bias can be viewed as the weight affected to a neuron connnected to the input, whose output is always 1. Neural networks

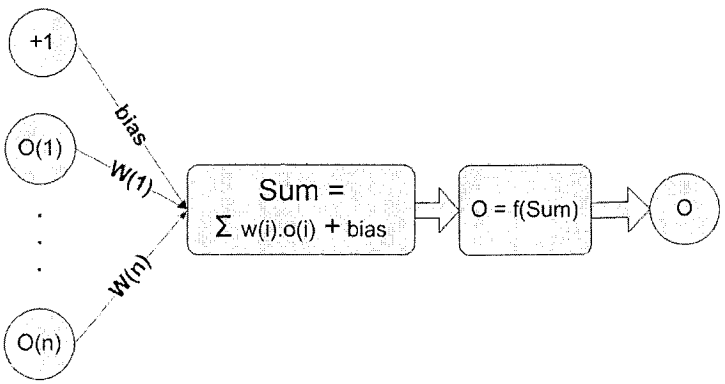

Figure 4.6: Neuron as computational unit

are well known as robust classifiers. Because of their resilience to noise, they are applied in different applications, which include automatic reading of postal codes, medical image classification [18],etc... As gestures are patterns subject to variations, their robustness was at the source of our incentive to use them for our classification problem.

4.2.2 Architecture

Yet, different from the aforementioned applications of neural networks, our problem presents a temporal aspect. Looking at the literature, we retained two architectural possibilities for handling time series classification with neural networks, which will be detailed in subsequent sections.

Besides the difference about how to handle the temporal aspect of gestures, both architectures nevertheless share common aspects. Both have an *input layer*, in charge of collecting the captured and preprocessed measurements. On

top of this layer figures a *hidden layer* whose role is to group the information to certain input neurons so that gesture models can be extracted easily. The connections between the input and the hidden layers rely on the notions of *finger* and *correlation factor.*

A finger may be understood as a group of neurons corresponding to a logical entity. If we consider joint signals as input data for the network, fingers will be groups of 3 neurons, affected to the joints of the same physical finger. If the tracker is also taken into account as input of the network, abstract fingers could be a group for the location (x,y,z) and a group for the speed$(\frac{dx}{dt},\frac{dy}{dt},\frac{dz}{dt})$.

Taguchi and Murakami in [32] used a correlation factor to decide how fingers should be interconnected. Their decision was based upon the fact that physical fingers are mostly affected by their closest neighbors. Therefore, hidden neurons will link together small groups of fingers, whose cardinality is the correlation factor. Experimentally they achieved their best recognition results with a correlation factor equal to 3.

The *output layer* contains as many neurons as gestures we aim at detecting and it is connected with the lower hidden layer forms a complete graph: every hidden neuron is linked to every output neuron. The results of output neurons can be viewed as detection probabilities.

TDNN

The *Time Delay Neural Network* approach intends to capture the temporal aspect of input by extending the input layer *spatially.* Instead of considering only measurements of a single time step, the input layer is composed of several *input units*, each of them holding the measurements extracted at a time step (see figure 4.7). The number of input units defines the network's time window.

Previously, we explained how fingers of the same input unit should be interconnected with the help of the correlation factor; this generates $C_{nbFingers}^{correlationfactor}$ hidden neurons that form a *hidden unit.* Consequently the hidden layer has as many hidden units as input units are present. Naturally, each input unit should be connected with its corresponding hidden unit. To model the influence that *previous* (temporally speaking) input units might have over a hidden unit, we decided to connect them with the same scheme.

The training of our TDNN is supported by a gradient algorithm known as the *backpropagation* algorihm, whose details are shown in [18].

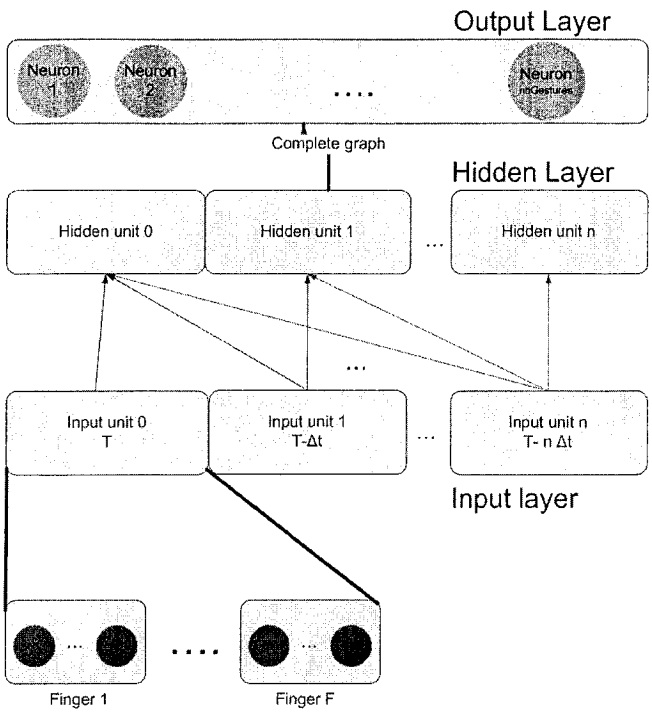

Figure 4.7: Time delay neural network architecture

RNN

Taguchi and Murakami have opted for a different approach: indeed they extended the previous TDNN architecture with a feedback mecanism: at each timestep, when the hidden layer's neurons output values are computed, they are at the same time stored in a *context layer*. Then, during the next time step, the stored values are reinjected in the hidden layer (see figure 4.8).

For the connections between the context and hidden layers, we decided to set them as a complete graph.

As the neural network is now a Elman *recurrent* network, the previous back-propagation training algorithm was adapted by Werbos [31].

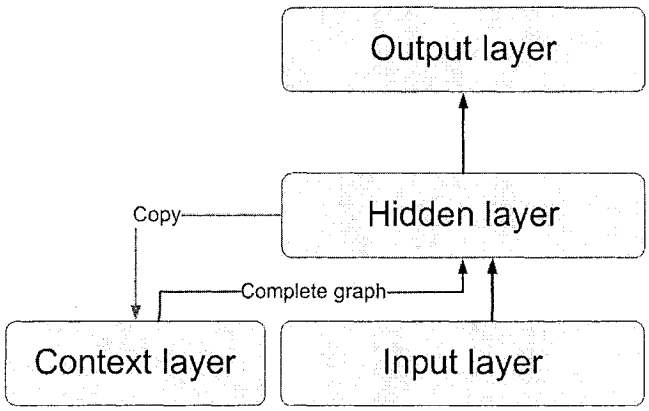

Figure 4.8: Recurrent network architecture

4.2.3 Issues

Neural networks are universal classifiers: they entirely partition the input space between the different gestures. Even if we decide of some detection threshold as Murakami and Taguchi did, we might run in some hypersensitivity issues: thresholds ensure that the input is *well* in a gesture region (see figure 4.9) but there is no guarranty that the input is near the gesture samples that were used for training. On the previous figure, when a geometrical shape is colored in yellow, it does not symbolize any meaningful hand configurations. When the network would classify an input as a gesture representation, the latter is figured by a circle, otherwise by a triangle.

Moreover, neural networks are not a good example of flexibility regarding our gesture set. Indeed, deciding of an additional gesture entails rebuilding and retraining a new network, with this latter often being time consuming.

4.3 Hidden Markov models

4.3.1 Introduction

Hidden Markov models are a probabilistic paradigm used successfully to detect temporal patterns such as *words* in the speech recognition area [19][20]. Without going into mathematical technicalities supporting this model, we can regard an

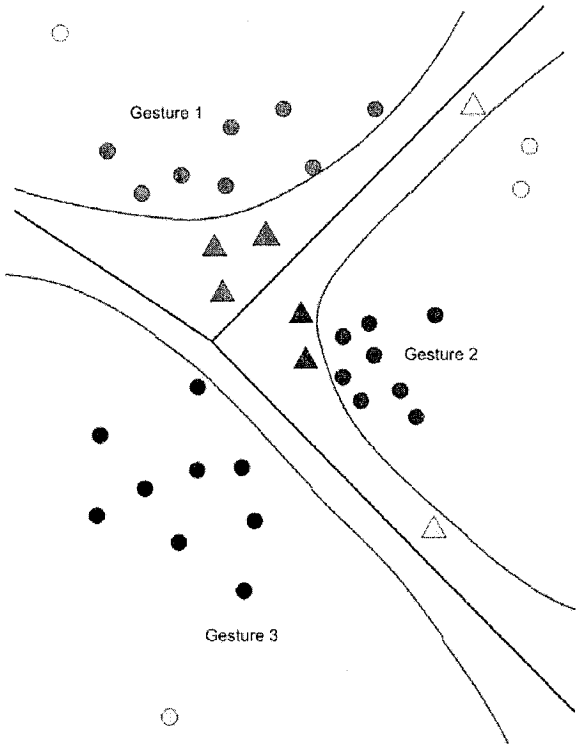

Figure 4.9: Neural network: effect of thresholding

HMM as being a set of interconnected *states*, each of them corresponding to a specific problem configuration at a given point of the time. The connections figure possible transitions between the states at successive time steps. Using the same weather forecast example as Rabiner in [40], possible states could be the seasons *fall*, *winter*, *spring* and *summer*, restricting the transitions to :

- *fall* $\rightarrow$ *winter*
- *winter* $\rightarrow$ *spring*
- *spring* $\rightarrow$ *summer*
- *summer* $\rightarrow$ *fall*

The example's transitions are deterministic. However, in more general models, where a state could lead to more than one state, transitions are probabilistic

usually noted as a matrix a:

- a_{ij} denotes the probability to go to the state j, being in the state i.
- Consequently, for each state i, the equality $\sum_{k \in states} a_{ik} = 1$ is true.

The expression of states is done through *observations*: continuing with the weather example, such observations could be *raining, sunny, snowing.* The fact that the model is in a given state is perceived when considering sequences of observations: for instance, during *summer* the weather should be mostly *sunny* with maybe a few *rains* but it is very unlikely to have *snow falls.* A possible observation sequence O might be $\{sunny$, sunny,rainy,$sunny\}$. On the contrary during winter a realistic observation sequence would be $\{rainy, snow, snow, rainy, rainy\}$. Thus the realism of observation sequences for each state is modelled probabilistically and usually noted using a matrix b:

- b is a N by P matrix, where N is the number of states and P the number of possible observations. b_{io} expresses the probability of perceiving the observation o while in state i.
- For each state i, we have the equality $\sum_{o \in observation\ set} b_{io} = 1$ true.

When trying to determine in which state of the model we currently are, we have to evaluate a given observation sequence probability. To perform this computation we lack one information: the *initial state conditions.* These conditions specify the chances of each state to be the one in which the observation sequence starts. We will note them as I; we also have the probability normalization condition $\sum_{k \in states} I_k = 1$.

4.3.2 Main HMM problems

- Observation sequence probability:
  Considering a HMM λ and an observation sequence O, we would like to compute the probability of O being generated by λ. Such question has been answered by the creation of two procedures namely the *forward* and *backward* procedures whose details are in [40]. The great advantage of these algorithms is the complexity reduction entailled: from an excessive $O(PN^P)$ to a more affordable $O(N^2P)$.

- Most likely state sequence:
  Given an observation sequence O and a HMM λ, which state sequence S is the most probable? The Viterbi algorithm detailed in [20] allows us to compute the sequence S maximizing p(O,S $|\lambda$).

- Training:
  Finally, once we have designed a HMM, we would like to twist its probabilistic parameters (transitions, observations, initial state matrices) so to make some observation sequences highly probable. Performing these modifications is known as *training* a HMM. To enhance the likeliness of some observations, an algorithm namely the Baum-Welch algorithm was developed, which can be consulted in [20].

4.3.3 Toward gesture recognition

As we have defined the different parameters describing a given HMM $\{\mathrm{N,P},a,b,Pi\}$, we have to adapt this paradigm to our problem. Previously, we have said that states correspond to a specific problem configuration at one point of time, therefore we intuitively thought that they reflect certain hand configurations.

What kind of observations do we have to capture instantaneous hand configurations (elementary postures)? After the feature extraction stage has been performed, hand configurations translate practically as a vector containing several measurements. This is an inportant difference with previous weather HMM model, in which observations were limited to a single piece of information (either *sunny* or *rainy* or *snowy*).

At that point, we have to decide whether to choose a multidimensional HMM architecture, which would take all measurements into consideration (observations are then vectors), or more a unidimensional HMM, which would require to summarize all measurements information in one piece. For simplicity sake, we opted for the latter strategy, adding a vector quantization process after the feature extraction step. Consequently, our observations set is the different *codewords* extracted plus a possible *noise* result from the quantization.

Also, in contrast to the previous weather example, no definite information is available about the state transitions. This time, the Markov model is *hidden*. We looked at some speech recognition frameworks for inspiration and noticed that the *left-right* strategy was very popular. This type of architecture models the

temporal irreversibility constraining speech units (which also exists in gestures) as an ordered sequence of states: transitions are only allowed from right to left (as in figure 4.10). In [26], Tanguay chose this sort of architecture for detecting mouse-driven gestures.

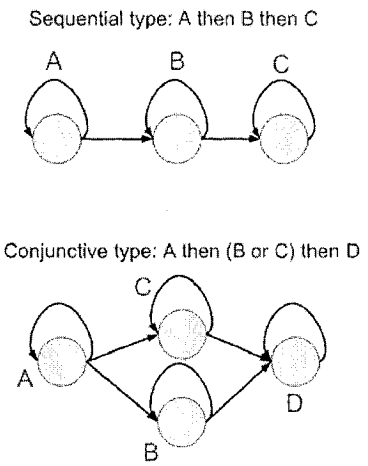

Figure 4.10: Two Possible *left-right* architectures

Viterbi based architecture

When considering using HMMs for recognizing gestures, we had an alternative:

- On one hand, we can build as many HMMs λ_i as gestures, each of them being specialized through some training with the *Baum-Welch* algorithm to detect a specific gesture. The main advantage of such choice is the flexibility it has with respect to the gesture set: indeed, if we want to recognise an additional gesture, we just need to buid and train a new HMM, the former ones not needing any modifications. The price for flexibility is the time spent in training.

- On the other hand, we can design a unique HMM in such manner than the most probable state sequence corresponding to an observation sequence indicates which gesture is the more likely to have been performed. If we use this idea, which relies on the Viterbi algorithm and a careful design

of the HMM, we would skip the training phase. Nevertheless, we lose the flexibility of the previous approach.

Eventually, we decided to try the second approach using the Viterbi algorithm. If G is the number of gestures, the HMM that we construct is composed from G parallel *branches*, each of them having its parameters set to make one gesture probable . This type of architecture has been successively employed in speech recognition to spot words [19] and gestures [37].

The challenge is to design the different *gesture branches* so that they favor a particular gesture kind excluding the other. As we hold states to represent a particular hand configuration, i.e. elementary postures, it appears logical to define a branch with as many states as different postures figure in the algorithm. For instance, if a gesture is encoded as $\{1, 2, 2, 2, 2, 3, 3\}$, a possible branch could be $S_1 \rightarrow S_2 \rightarrow S_3$. Each state S_1, S_2, S_3 favoring a particular observation; in our example S_i would make the i-th observation most probable.

This branch model does not take into account the *variety of samples encodings* and the state duration problem. Indeed, when observing samples encoding, we may find some variations in elementary postures used, in the order of the elementary postures and in their durations:

- $\{1, 1, 1, 2, 2, 3, 3, 3, 4, 4\}$: reference sample.
- $\{1, 5, 1, 2, 2, 3, 6, 6, 4, 4\}$: presence of different elementary postures.
- $\{2, 2, 1, 1, 1, 2, 2, 3, 3, 4\}$: order change occuring.
- $\{1, 1, 2, 2, 2, 2, 3, 3, 3, 4\}$: variation in state durations.

Reasons for differences in elementary postures used in encodings are twofold: some codewords in the vector quantization may be so close that they could be easily mistaken. Another explanation would bring up some human factor: when recording several samples of a gesture, we unconsciously use different but close hand configurations. For the performer, a unique gesture has been made but objectively it may not be the case: when doing the *fist* sign, the thumb could be put on top of the index and middle finger or just beside them.

Differences in state order may occur in certain gestures similar to the *scissors* one, which are difficult for the user to perform many times similarly. When *scissoring* it is hard to open and close one's fingers a definite number of times.

Finally ,the state duration difference are easily explained by some execution speed variation between samples.

Gesture branches

In order to construct a branch of our HMM , we ask the vector quantization stage for a gesture model. Observing the model, for example {1,1,2,2,2,3,3,3,3,1,1,1, 1}, we can deduce the stages of the gesture (*1,2,3,1*) that will be our states. The branch will be composed of as many states as different successive stages there are in the model.

As we can observe on the previous figure 4.11, states are provided with self transition loops. We have to set these loops' probabilities so that they reflect the length of stages. Observing the previous model example, the last stage, with *1*s, self transition probability should be higher than the one of the first stage. Thus we decided to affect to each state a self transition probability equalling the relative length of the stage it represents (relative should be understood as the ratio of the stage length over the model length). The probabilistic aspect of state transitions may support us to some extent in our quest to cope with distortion generated by variations of execution speed: the different stages proportions and time window are fuzzily specified and not hard coded as when using a metric.

For each state, we need to define its observation probabilities. In our design, a state corresponds to an execution stage of the gesture model. As execution stages of a gesture model are defined as a sequence of a unique codeword c, we could restrict all observations to c by setting its probability to 1. However, such radical choice is not judicious since it threatens to nullify the probability of sample encodings with slight variations with respect to the model.
Instead, we decided to consider the totality of sample encodings for the observation probabilities: table 4.1 demonstrates the alternative procedure. Indeed for each state an observation probability is set to the average of its occurences between the different samples.

At this point, we have set the observation probabilities by examining our samples encodings. The vector quantization process having been created from the gesture set, it is very unlikely (at least using the LBG algorithm) that some noise result occurs in the encodings. Yet, facing a user performing our gesture in real time, we must expect some significant variations between his gesticulations and the ones that were recorded. Those variations when too important would

Gesture model	1,1,1	3, 3
Sample 1	1,1,2	3,3
Sample 2	1,1,1	3,3
Sample 3	1,1,1	1,3
Sample 4	1,2,1	3,3
States	State 1	State 2
Observations probabilities	1 → 0.83	1→ 0.125
	2→ 0.17	2→ 0
	3→ 0	3→ 0.875

Table 4.1: Observation probabilities estimation

be encoded as noise and if we do not set some noisy observation probability, every branch of our HMM would have the same probability 0.

To incorporate some noise robustness, we considered a noise percentage factor P_{noise} (ranging from 0 to 1): the observation probabilities b_i are modified as $b_i(1\text{-}P_{noise})$.

When about to use the Viterbi algorithm to see what state sequence is most probable to have generated a given observation sequence, the initial state probabilities are needed. As we are interested only in *branch* state sequences, i.e states sequences based on a complete branch, we set the initial state probability of the starting state of our HMM design to 1.

Now that every parameter of our HMM has been set, we may use the Viterbi algorithm. Looking at the possible state sequences yielded by this algorithm, we can see that some of them may not correspond to a gesture branch in its integrity: if S_0 is our starting state, S_1, S_2, S_3 the states of a gesture branch expressing 3 distinct execution stages, we might obtain a sequence similar to S_0, S_0, S_1, S_2, S_2. In that case, since the last stage of the gesture has not been observed yet, we consider that the observation sequence does not express any gesture. As a conclusion, a gesture is only deemed recognized when a branch is completely present in the Viterbi resulting state sequence. Additionnaly, as the transition probabilities between states of different branches are null, there can be no ambiguity about the branch.

One feature of HMMs that has been used for speech recognition is their ability for segmentation [19]. To exploit this possibility, we renounce to a full

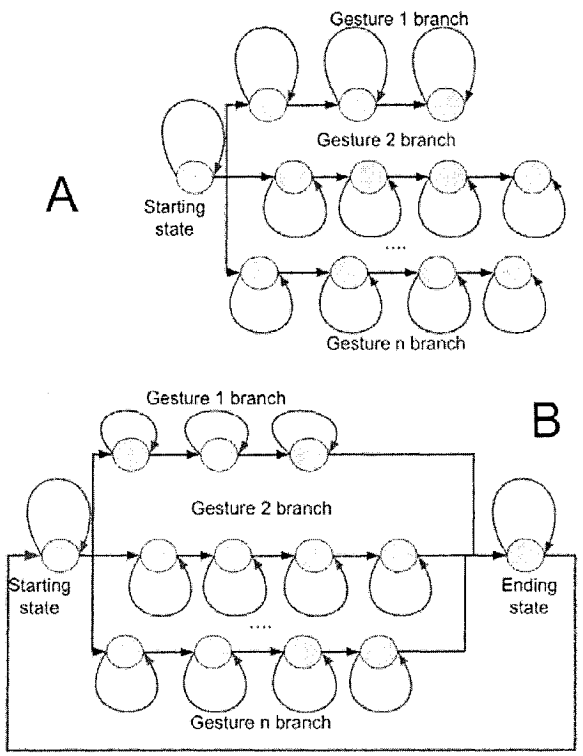

Figure 4.11: The 2 possible Viterbi architectures: (a) without ending state ; (b) with ending state.

left-right architecture by connecting all final states of every gesture branch to an ending state, which is also connected to the starting state as it is shown on figure 4.11.

4.3.4 Discussion

In this section, we will underline some aspects of our HMM framework that make it particularly appealing to deal with a GR problem. Foremost, this approach allows us to consider time windows of arbitrary lengths, permitting to deal with longer or shorter gesture execution times. This overall flexibility for the time span, within which gestures are performed, is also present *locally* at the gesture stage levels by the self transition loops. This resilience for gesture distortion can

be also found at the execution stage level in the template matching approach when using the DTW metric, however the HMM approach handles the noise problem more naturally.

Moreover, in a HMM framework, in which a final state is included, the segmentation problem is also tackled: whenever a state sequence resulting from the Viterbi algorithm presents a *come-back* from the ending state to the starting state, a distinction between two gestures has been done.

Finally, an enjoyable aspect of HMMs is their high level modelling of gestures through gesture stages.

Chapter 5

Gesture interface

5.1 Generalities

The GR framework was developed in c++ so that it could be eventually compiled as a *dynamic link library* (**dll**), which could be directly used in c++ or Java applications to provide some gesture interactions. The choice of c++ as programming language was motivated by the capture devices whose drivers are also written in this language.

Due to the high dependency of the recognition results on the set of gestures, we renounced creating an API, whose recognition procedures should be used hard coded in some application code. Indeed, such procedures would necessarily hard code the values of different parameters, notwithstanding their optimality. Instead, we opted for a recognition framework, in which settings intervening in the gesture detection process could be chosen dynamically through a *graphical user interface* (**GUI**).

5.2 Implementation

The implementation consists of auxiliary classes, each of them performing one recognition stage, and a main *Gesturon* class in charge of coordinating every recognition step. Besides, the framework's GUI is contained in a single source file (GUI.cpp).

5.2.1 Auxiliary classes

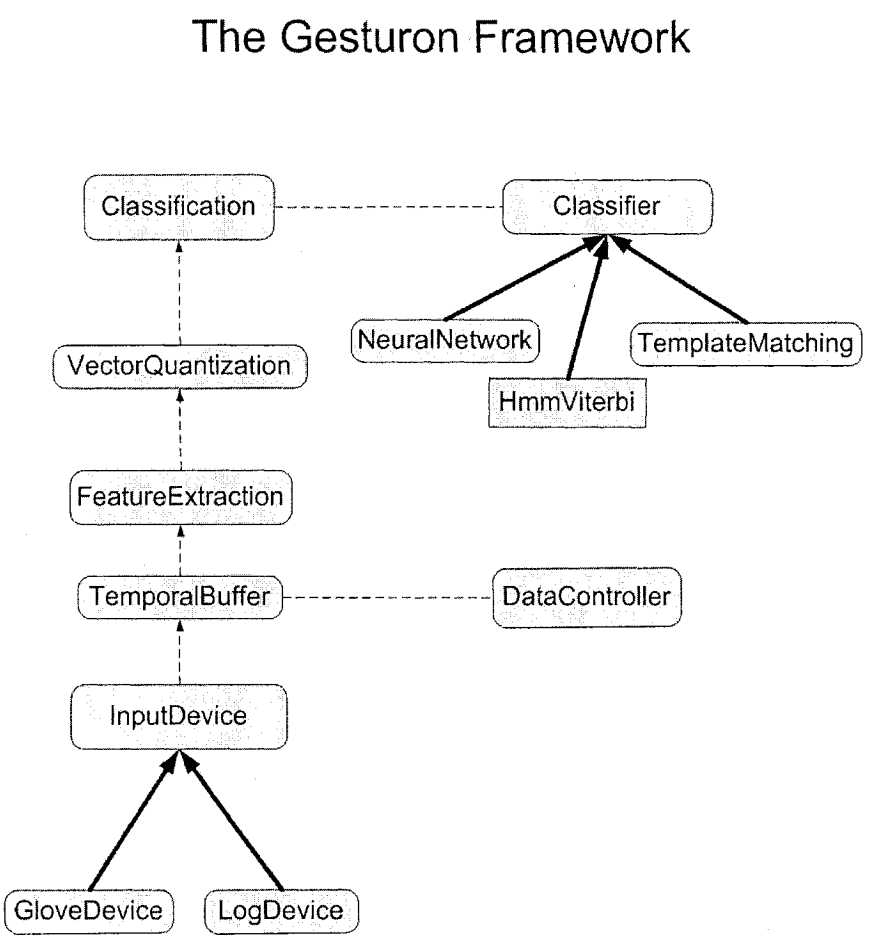

Figure 5.1: The different components

Data retrieval

The first step in the recognition process consists of retrieving the various joint measurements characterizing the hand configuration. This task is supported by the abstract class *InputDevice*. In the case of real time recognition, we refine this class in *GloveDevice*, which will communicate with an instance of the

cyberGlove^{TM}. An alternative way of testing the recognition framework is to load a *log* file, in which a precise sequence of gestures has been recorded. In that case, there is no longer need to communicate with the cyberglove^{TM} and the class *LogDevice* should be chosen.

Once the measurement capture strategy is specified, the feature extraction stage requires us to keep track of them for a definite period of time (time window) in order to calculate some temporal features. To meet this requirement, the *InputDevice* instance is coupled with a *TemporalBuffer* object. The *TemporalBuffer* creates a circular buffer for storing all joint successive measurements. This coupling has the effect that whenever the input device is updated, it also actualizes the buffer.

Yet the buffer raw data are not directly exploitable for extracting features: we need to perform the normalization preprocessing step. The class *DataController*, in charge of recording, storing, retrieving *Gesture* objects from the hard disk, will load the necessary *user log* for the normalization process. In addition, the *DataController* instance is coupled to the temporal buffer so that after each update, all buffer measurements are normalized.

Representation

Now that we ensured that the data buffered are normalized, we can use the two following classes to represent gestures:

- The *FeatureExtraction* class, which is responsible for extracting features depicting the current hand configuration. The result of this stage is a feature vector. When the user did not specify any features, the feature vector is composed of the latest normalized joint measurements, otherwise it contains first the temporal feature value for each joint and then the static features.

  Regarding temporal features such as *mean* and *variance*, we encountered an issue due to the finite length of gesture records. Indeed if the time window is t (in time step), G $= \{g_1, g_2, .., g_n\}$ a gesture record of length n, to extract the temporal features corresponding to the first step g_1, we would need to know all t *previous* hand configurations (g_{2-t},..,g_0,g_1).

  Obviously, extending the time during the recording would not be a good solution since the first steps would describe hand configurations that are

no components of the gesture itself: this would certainly entail some articulation differences between samples of the same gesture.

To overcome this lack of history and adress this temporal feature problem (**TFP**), noticing that users tend to have their hand poised for some time in the initial configuration of a gesture, just before performing it, we decided of a *padding* strategy (see figure 5.2.1: in preparation of the feature extraction, we fill up the buffer with the g_1 measurements. Then, we update the buffer with the different elementary postures g_i composing the record.

- For some classifier, namely the HMM one, quantizing the resultant vector is necessary. For this purpose, we can use the *VectorQuantization* class, which will allow us to construct a codebook and encode feature vectors.

Classification

From an implementation point of view, we decided to top each possible classifier by a *Classification* class that will take care of the supervision when needed . This class is coupled to some *Classifier* object. Actually, classifier objects are not instances of *Classifier*, since it is an abstract class but of 3 classes that inherit it:

- A *NeuralNetwork* class:
  This class is dedicated to the instanciation of a neural network framework according to the parameters chosen by the user. The neural net data structure itself relies on more elementary classes:
  - a *Neuron* class embodying the different computations previously presented. An *input* value is set, then passed to an *activation* function, whose result is stored in a *output* value parameter.
  - a *Layer* class, which is refined in *InputLayer* for the first layer of a neural network does not obtain its input values from the computations of neurons but from some external utility (*VectorQuantization*)
  - a *Connections* class specifying how connections should be established between two layers. This class is also responsible for the propagation (respectively backpropagation) of the output (resp. error) results from a layer to another.

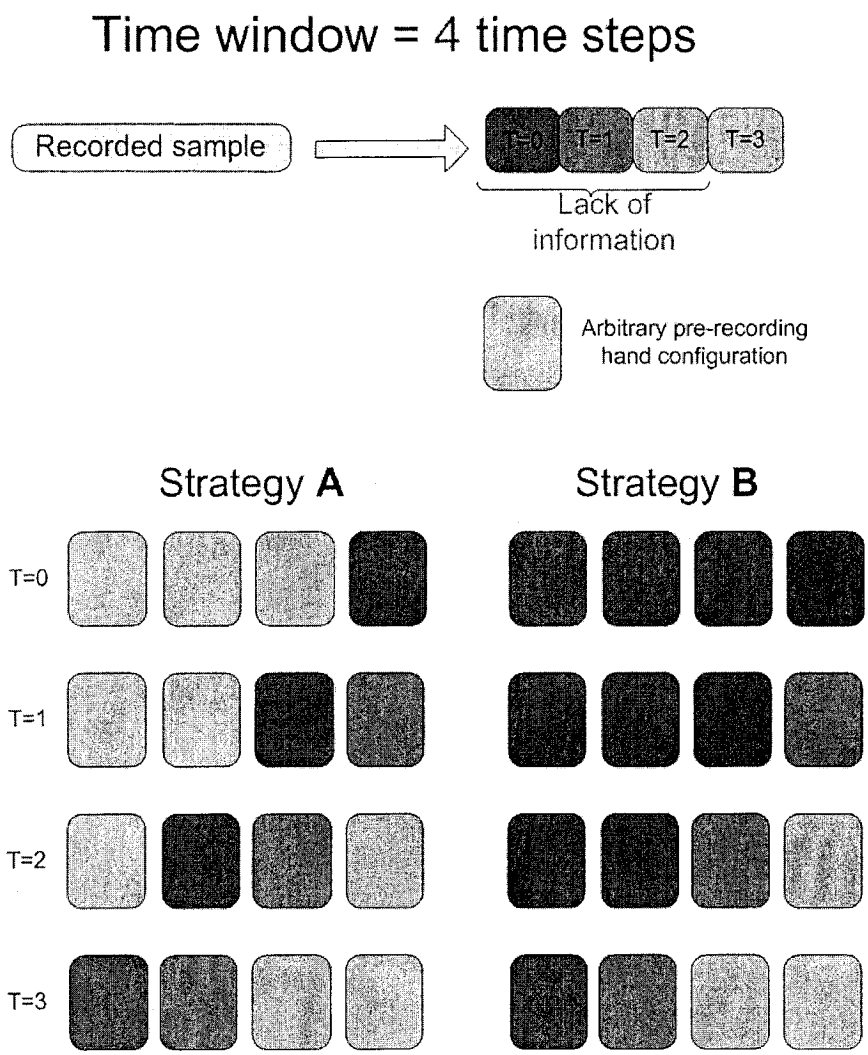

Figure 5.2: Temporal feature extraction problem

- a *Connections* class. This storage class contains the weights of the links connecting neurons. Moreover, it is in charge of propagating the results forward and the errors backward.

- A *Hmm* class:
  This class provides the bulk of the HMM design. It provides a method

to evaluate an observation sequence's probability and its corresponding most probable state sequence (Viterbi algorithm). Yet, no logical view is offered: states are not coded in structures, the transition, observation and initial state probabilities are stored in raw matrices.

As the low level of the previous class prevents us from using it with convenience for gesture recognition purposes, it is topped by the *HmmViterbi* class. This latter elaborates a HMM with respect to the design previously presented based on the gesture representations obtained from the vector quantization.

- A *TemplateMatching* class:
  This class performs the template matching task for a selected metric.

5.2.2 The *Gesturon* class

The *Gesturon* class is the main class, from which the different recognition parameters can be set. To that extent, it acts like an interface for the programmer. Thus the GUI of the recognition framework is entirely connected to the various parameters through this class. Currently the GUI (figure 5.3) has been developed with *GLUI* [8], an open source c++ graphical library:

5.3 An application example

Our gesture recognition work was demonstrated at the Learning Object Repositories Network (**LORNET**) conference [3] by developing some file system navigation application (figure 5.3). This application, written in Java3D, creates a VE, in which some types of files (directories, documents, music samples, videos) are represented by different 3 dimensional shapes. The purpose of this program was to demonstrate how interaction within a VE could be supported by a gesture interface.

During the development of this application, in addition to recognition issues, we faced new challenges related to user interaction. Indeed, as briefly mentionned in the introductory chapter, additional parameters, normally handled by a *manager*, should be taken into account so that gestures could be properly exploited for interaction.

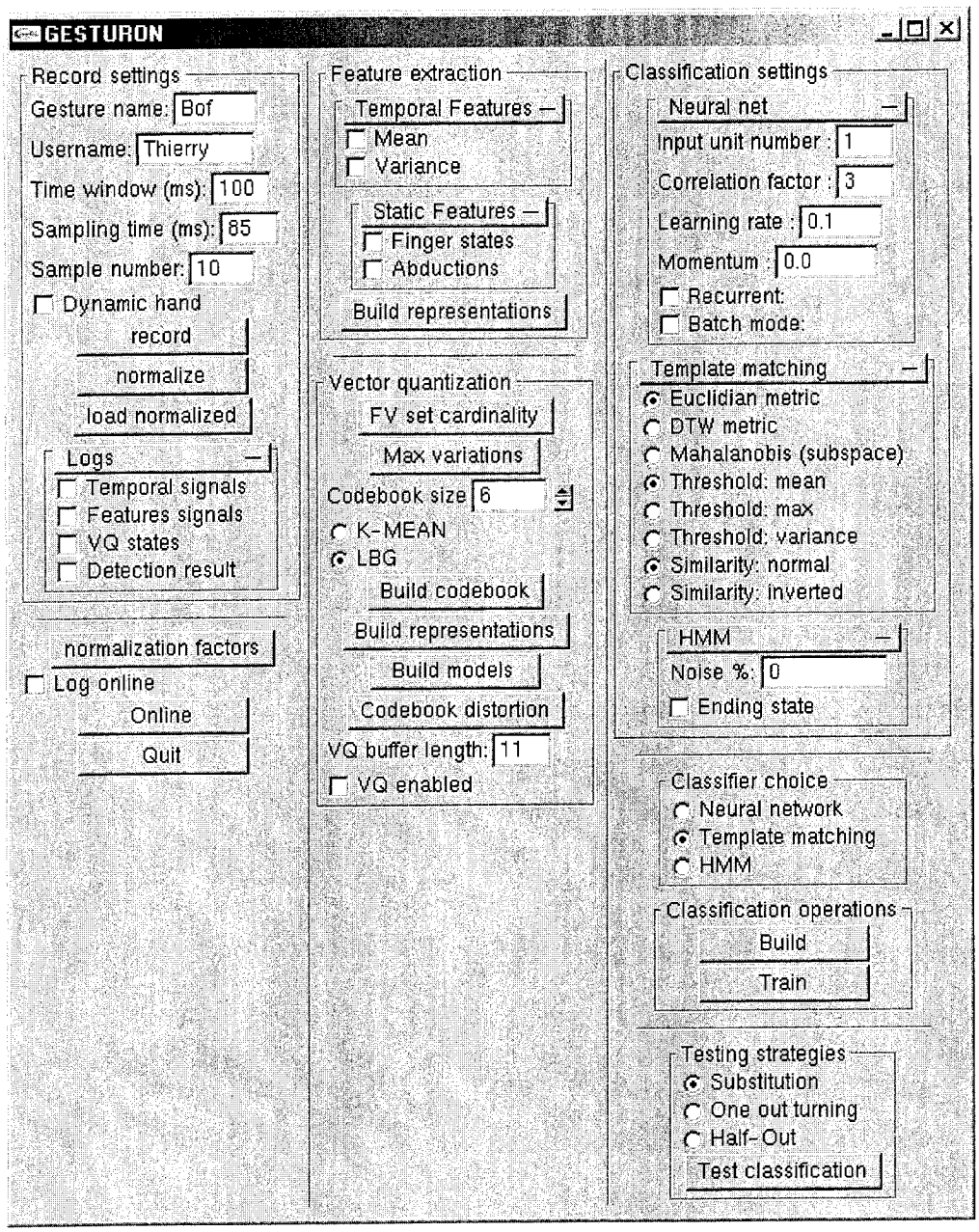

Figure 5.3: The GLUI based framework

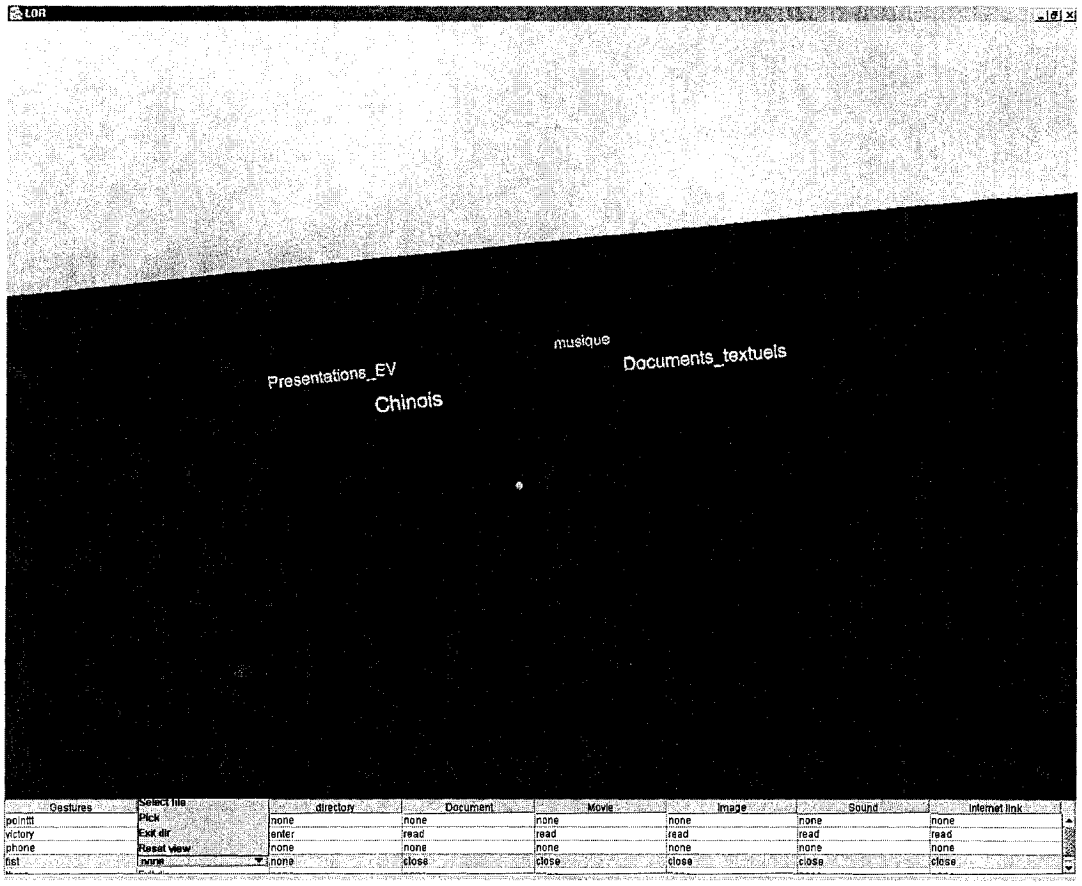

Figure 5.4: File system gesture based browsing application (main view)

For instance, the recognition rate used for detecting gestures (at least 10 Hz) is far higher than the user's response rate. If no measure is taken, a unique gesture execution usually ends up in a series of identical gesture signals. Some other issues we encountered were more about the semantic values of gestures with respect to their application context.

5.3.1 Architecture

File Objects

Our file system navigation application relied on a hierarchy of *file objects* (see figure 5.5). At the top of the hierarchy is the class *FileObject*, which is responsible for the *visual* parameters of the corresponding virtual object in our scene such as the location, orientation, geometry, but also for *logical* parameters (file name, directory location,etc ..). This logical information is stored to match a visual representation and an actual file stored on the disk. The *FileObject* class, only existing to provide basic functionalities common to all files, has been defined as an abstract class.

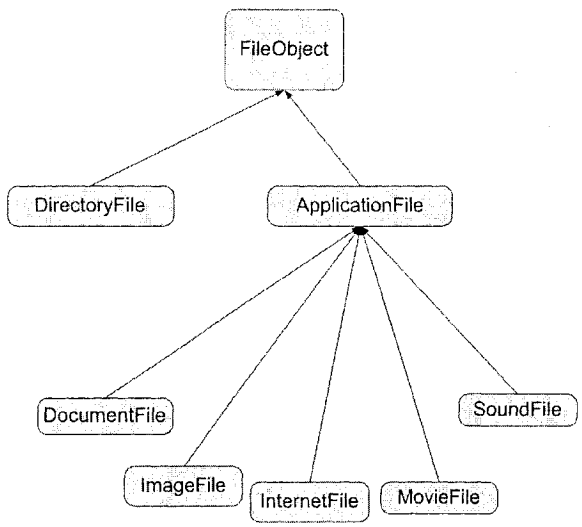

Figure 5.5: The file object hierarchy

In a second time, we distinguished two types of files: *directories* and *application* files. The former type dealt with the *DirectoryFile* class and allows us to enter, leave a given directory, retrieve all file objects present inside. *ApplicationFile* instances are to be understood as normal files that can be executed (see classes derived) by some programs. For example, when we want to open a **.doc** document, the *Microsoft Word*^{TM} program is called whereas for an **.mp3** musical file, *Winamp*^{TM} is preferred. The different subclasses of *ApplicationFile* are dedicated to specific sorts of files (audio,multimedia,web links, documents) and

it discriminates them according to their extensions. To that extent, a **.txt** would be executed using *wordpad*^{TM}, whereas a **.pdf** would call *Acrobat Reader*^{TM}.

Selection

Besides the file objects, our application uses a *Navigation* object to control the scene view. This class is responsible for the orientation of the view panel (when the user wants to look in another direction), and for selecting a virtual object. The user select an object by performing a *picking* gesture (typically using the *pointing* sign), which after interpretation finds the object using the Java3D picking facility: a virtual ray is created, originating from the center of the view and whose direction is orthogonal to the view panel. If this ray intersects some geometries, the object selected is the one closest to the user. When the selection is successful, the file object's name in the scene changes color from white to red.

This selection strategy entails that in order to select an object, the user must move himself right in front of the latter: the object should appear in the center of the window, possibly not far, lest it would be missed by the ray if the view plane is not orthogonal to the [object-viewCenter] direction] (see figure 5.6).

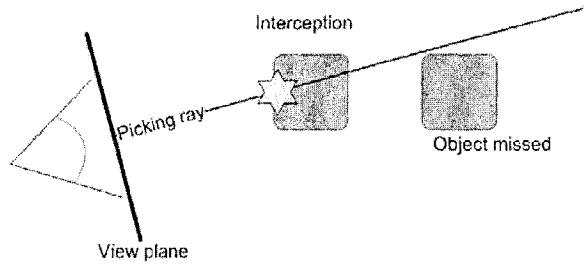

Figure 5.6: Ray deviation problem

At that stage, we presented how a *geometry* may be selected, yet the user wants to have access to the corresponding file object (with both logical and geometrical parts). For that purpose, when creating a file object, we recursively linked every node composing its geometrical appearance with a reference to it. When a *Shape3D* object is selected, we use then this reference to backtrack to the *FileObject* instance.

Navigation

The VE scene that we built for this application is relatively coarse: file representations were grouped near the origin, a uniformly colored horizontal plane symbolizing a ground, the whole being enclosed in a textured sphere figuring a sky. As a matter of fact this environment by its uniformity, when one is not close to the center, makes the user easy to be lost by its absence of perceptual cues.

In a 3 dimensional environment, navigation is based on two different types of motion: *translation* and *rotation*. For simplicity's sake and in order to spare the user's sense of orientation, we decoupled both kinds of motions. Directions of displacements are obtained from the tracker device after having defined a user referential (see section 2.1); we had then an alternative:

- On the one hand, we can base the movement in the VE on the hand motion: when the hand moves right, the user view is moved right and so on. However, this strategy is not practical for three reasons: first the range of the tracker emitter is limited (3 feet) and when moving the hand quickly reaches the range limits. Second, if we desire to move in a constant direction, the user will instinctively engage in some back and forth hand movements to remain within the range of the emitter and if the gesture for moving is still done when recentering the hand (which is probable), this intended movement in the VE will be affected. Finally, motioning is tiresome and more difficult to control than holding one's hand steady, which brings us to the other possible choice that we eventually retained.

- On the other hand, instead of motioning in the direction of the movement, the latter will be indicated by the hand's current location. Around the center of the user referential, we defined a neutral sphere (the user decides of its radius), inside which no motion will be generated. On the contrary, when the hand is outside, we determine the coordinate with the greatest absolute value. The direction (*right,front or up*), corresponding to this coordinate, will be the one of the movement in the VE.

Now that the direction of a translation or a rotation is decided by the user's hand location, we still need to define the movement's amplitude. Since in computer applications, motion continuity is only an illusion resulting from a proper choice of displacement's steps and repetition rate. The smaller the steps are, the

better is the continuity impression. Yet, at a constant repetition rate (our case), we need more time to do cover a given distance. Choosing too big steps may degrade considerably the motion continuity visually and, as it limits the granularity of displacement, may also prevent us from positioning ourself correctly in the VE (thus forbidding us to select some file objects). When we decided of some tradeoff for the movement steps, we were conscious of that our navigation would be very basic. Indeed, our way of handling displacement parameters (the most basic one) is far from providing all refinements as classic input devices managers would: for instance, the mouse manager allows the user to choose precisely the speed and acceleration of the cursor.

Actions

Each time a user performs a gesture, we face the problem of its interpretation. Hard coding the semantic value of each gesture is feasible but not recommendable, since we would have to consider as many gestures as different possible actions. The reasons about the negative effect of an increase in the gesture set size are twofold: first, it complicates the recognition process and second, we have observed experimentally that for users not accustomed to sign languages juggling between more than 5 signs to execute precise actions, proves to be uneasy.

As a consequence we decided to use a limited set of gestures, which could be interpreted differently with respect to the context of their execution. For this purpose, we defined an interface (*GestureListener*), which objects of the scene susceptible to gesture interaction should implement. This interface specifies among other a method *gestureDetected* taking a *GestureEvent* object and an action index as arguments. Gesture events are fired when the recognition framework detects a gesture; the corresponding objects contain a gesture code, i.e. an index in our gesture set, and secondary information such as the hand location at the time of the gesture execution. The response of a *GestureListener* object, when activated by a gesture, is contained in the code of its *gestureDetected* method.

In the case of a hierarchy of *GestureListener* objects, in which it might be useful for subclass instances to share the same behaviors as their parents, the inheritance present in the implementation provides a good support. Indeed, we thought of embedding in each class of the hierarchy an array of *actions*, which

would be updated as long as we go down in the inheritance. For instance, if the *FileObject* class action array is set to {**copy,paste**}; in the *ApplicationFile* child class, the constructor will append to this array **execute**. In parallel, the *gestureDetected* may reuse the some already coded in parent classes actions by calling the same method defined in their parent, before a switch mechanism:

```
void gestureDetected(int action,GestureEvent evt){
   super.gestureDetected(action,evt);
   switch(action) {
    case NEW_ACTION_1:
         ...;
         break;
   }
}
```

The *context awareness* flavour that we previously presented can be seen in the lower part of the figure 5.3: in the visible two dimensional array, columns (respectively rows)represent file object types (resp. loaded gestures). Each cell of the array displays the available actions through a drop down menu. All action names are obtained from the *GestureListener* objects by using a *getAction()* method returning their action array. When the user attributes a specific action of a *GestureListener* object for a given gesture, we update a two dimensional *gestureActions* array of indices (following the same column and raw convention). As a gesture is performed to act on a reactive object in the VE, its *gestureDetected* method is called with the action argument being:

$$gestureActions[\text{gestureType}][\text{fileObjectType}]$$

Readers somehow familiar with Java may notice that the vocabulary we employed and our approach, aiming at interpreting gesture events with respect tot their application context, remind them of the Java *events* and *listeners*. In fact, the Java language, with his flexibility in processing different inputs (mouse, keyboard) in a similar manner, was our main source of inspiration for handling gestural interaction. When trying to emulate further this Java approach about interaction management, we became aware that the present stage an ingredient was still missing. In Java, if we create a window and attach some mouse listener to it in order to turn it responsive to mouse actions, the window will not necessary react to mouse generated cues. This apparently paradoxal absence of

response can be easily explained by *management* considerations: most applications, once launched, would be terminated by "ALT+F4" signal, yet when the user does this combination of key strokes, he only intends to get rid of the one, on which he is *currently* working on. The "currently" in that context means that the application has received the user's attention or *focus*.

Focus management

The previous paragraph implied that every gesture was applied **on** an object. When considering some actions that are translated by transitive verbs, this fact is obvious: *copy* text-file. For actions whose verbs are intransitive (*move* right, *turn* left, *pick*,...), there is apparently an absence of targeted object. When diving in implementation details, we can notice that in practice all gestures act on some objects. Indeed, these objects may be part of the graphical scene (file objects) or some abstract entities needed by the application. The previously quoted intransitive verbs actually work on the *Navigation* object.

From the previous remark, we can deduce that the physical virtual object selection by picking is insufficient, since not all *GestureListener* instances have visual representations. To address this issue, we came up with the concept of *focus*: only *GestureListener* objects having obtained the focus, i.e. activated, will respond to gesture interaction. When the application is running, we keep track of activated objects by storing their references in a focus list. Once a gesture event is fired, we hand it out to all members of the focus list.

Focus management is one of the thorniest aspects of developing and amounts to a large part of the application's successful usability. As we very succintly touched on this part, we will only present a few related difficulties that we encountered. Certain abstract objects should always be activated: without the *Navigation* object enabled, neither roaming through the VE nor selecting a file object are possible, reducing therefore the application to a mere static view. The selection of file objects, through the Java3D picking mecanism, activates them; at that stage, one should be cautious that a unique *GestureListener* object's reference be stored more than once in the focus list. Objects having received the focus have become sensitive to actions triggered when gesticulating: this sensitivity raises the problem of deciding *when* an object should lose its focus. This issue is rooted in the fact that actions may be repeated with more or less tolerance:

- Actions similar to *translating the view, moving a file object virtual representation* may be repeated several times without having dramatic consequences. In some case, action repetition is even an advantage: when the user wants to move an object's virtual representation, the smaller the translation pace, the more translation action will be needed. In that case, if the object lost its focus after a single translation and would have to be re-picked to continue its journey, it may very quickly exhaust any user's patience.

- Conversely, for some actions repetitions are not tolerable. The *pasting* procedure, which recreates in front of the user a geometry similar to a selected object, examplifies this case well. Indeed, repetitions, possibly due to the slowliness of the gestural input compared to the recognition rate, would cause numerous *clones* of the original file object geometry to appear. The disturbance may not only be visual: for instance, if the *execute* action is called more than once on an *ApplicationFile* instance, it will result in multiple calls of the same program, which might be desastrous (experimentally we had some crashes for this reason with certain multimedia programs).

  Thus, when dealing with similar kinds of *one time* actions, it is safe to remove the targeted object from the focus list.

Both previous extreme cases of repetition tolerance are in no measure exhaustive; some more elaborate applications might generate some complex action scenarios.

Another issue when managing focuses occurs when one faces several activated objects subscribing to a common gesture (having a precise action to perform for this gesture). When the corresponding gesture event is generated, all objects would be simultaneously affected, yielding exotic results. We acknowledge that this issue was completely left aside, for this purpose we judiciously set the (*file type,gesture type,action*) correspondences in a favorable configuration when starting the application.

Chapter 6

Recognition Evaluation

Evaluating results of a gesture recognition framework is quite difficult for two main reasons:

- The problem complexity greatly depends on the set of gestures we consider: capturing 2 very distinct signs is much easier than a set of 5 dynamic gestures presenting some articulation issues.
- The different stages involved in the recognition process can be parameterized in many ways, influencing each other.

6.1 Test gestures

In order to show the framework's behavior, we restricted ourself to the following gestures that were recorded using a sampling rate of **11.75** Hz and a time window of **1** second. For each gesture, we asked the user to record **10** samples:

- The **gun** static gesture.
- The **victory** static gesture.
- The **scissors** dynamic gesture. This gesture is usually done by people intending to cut something. The dynamics of this gesture chiefly rely on the index-middle finger abduction joint and to a lesser extent, on other joints because of physical dependencies. When performing *scissors*, problem articulation may occur, for the hand shape alternates between the two

previous postures. The *scissors* gesture can be qualified as periodic since it is characterized by alternance of two postures.

- The **openPalm** static gesture. In this posture, the hand is flat with all fingers straight and abduction joints open.
- The **fist** static gesture.
- The **C** ASL sign.
- The **grabbing** dynamic gesture. The user starts in an *openPalm* configuration and progressively curles all fingers to end up with *fist*. We can notice that another articulation problem is likely to occur with respect of the three precedent postures. Besides, this gesture is transitory: normally no repetition of a hand configuration occurs during its execution.
- The **quoting** dynamic gesture. The hand starts in the **gun** configuration, then the user bends and extends both index and middle fingers repeatedly.

The different testing sets used for evaluating the different stages of the GR process were the following ones:

- $\mathbf{A} = \{$ *gun, victory, scissors* $\}$
- $\mathbf{B} = \{$ *openPalm, fist,C sign, grabbing* $\}$
- $\mathbf{C} = \mathbf{A} \cup \mathbf{B}$

6.2 Representation

To observe how does our representation stage perform, we decided to work on the *C* gesture set.

6.2.1 Padding strategy effects

In order to compute temporal features for the first measurements obtained, we applied the previously explained *padding* strategy. Postures by definition are not affected, for the hand is poised therefore time invariant. On the contrary, for dynamic gestures, padding the temporal buffer to compute the *mean* and *variance* features has visible consequences that can be observed on figures 6.1 and 6.2.

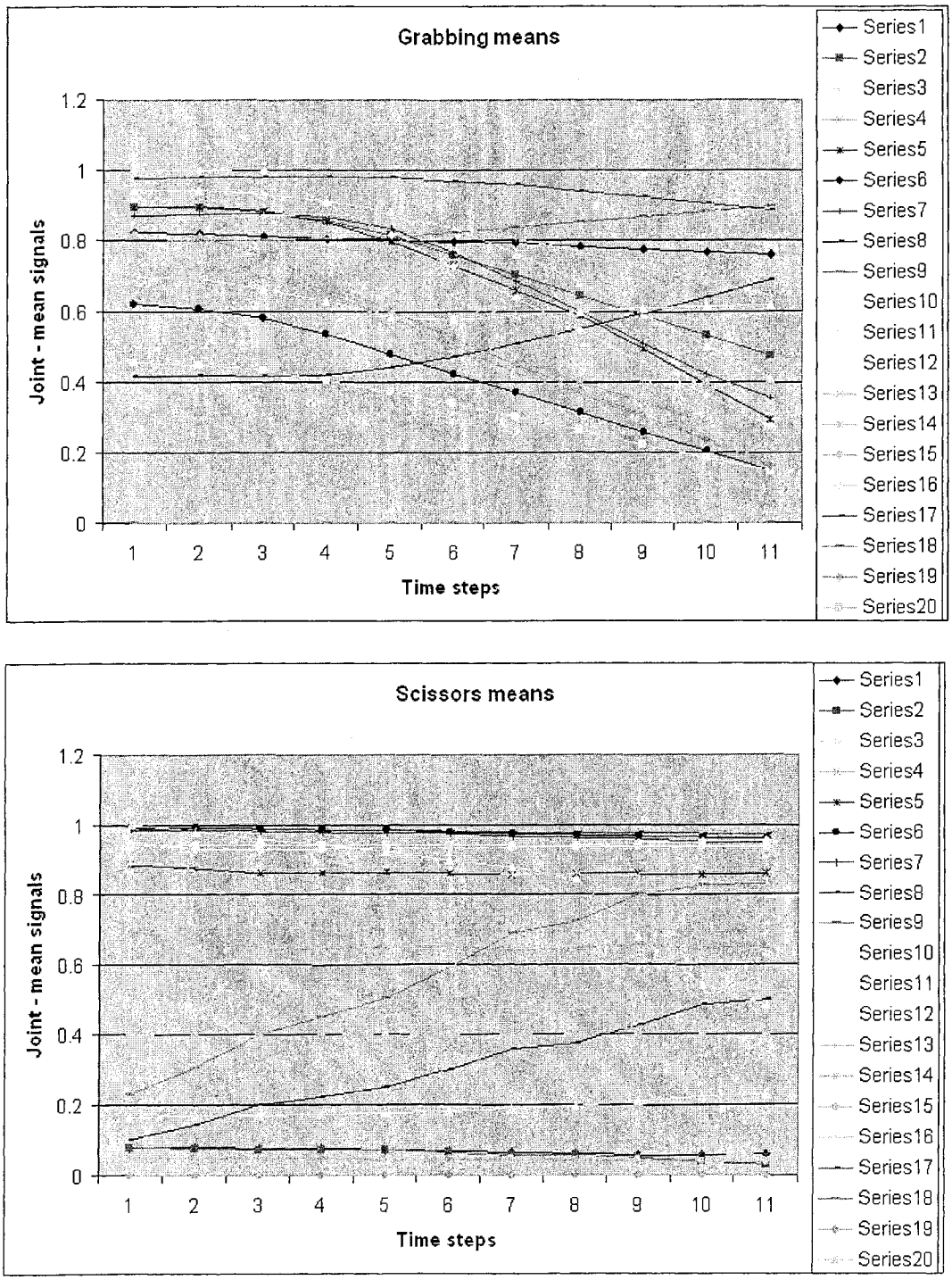

Figure 6.1: *Padding* effect on *mean* feature

As we can see for both dynamic gestures *scissors* and *grabbing* on 6.1, using the padding strategy tends to make the mean signals sluggish for the first time steps: for the 5 first time steps (almost half of the time window), the joint mean feature values correspond roughly to the ones of starting posture. Moreover, this sluggish aspect may be misleading, for the scissors gesture is periodic by nature (repetition of changes of the thumb-index abduction joint) and thus should have all its mean feature values almost constant (which is not the case).

On figure 6.2 displaying variance signals, we can see that all start at the value 0 and gradually increase for the joints having a dynamic behavior (only 3 for the *scissors* gesture). Although this initial anchoring at 0 is problematic, it does not prevent discriminating dynamic gestures from static ones. Indeed signs variance values do not exceed 10^{-2} (worst case due to some involontary tremor during recording), usually being around 10^{-5}, whereas dynamic gestures would see the variance signals of their dynamic joints over 2 10^{-2}.

6.2.2 Vector quantization

To construct the codebook necessary for quantizing feature vectors, we tried the two previously explained methods *k-mean* and *LBG*. As we decide a codebook size, the LBG method, due to its algorithm specifications, will consider the next power of 2 for a codebook size. Since our adaptation of the *k-mean* algorithm is non deterministic, for during the seeding phase we pick random feature vectors to be cluster centroids, the indices occuring in a gesture encoding might vary between executions: the *gun* posture might be first encoded as a sequence of 0, then after the codebook has been built with the same parameters as a sequence of 2.

Reading the next tables, one should not attach any signification about the values shown in the encodings, since these are mere indices denoting the different codewords (see section 3.5). What is of interest to assert the quality of the gesture representations, is the recurrence of these indices in and among samples. Each encoding displayed in the tables is composed of 11 time steps as the sampling frequency was a tad less than 12 Hz. The vector quantization process being applied on top of the feature extraction step, we specified in the tables' captions what features the feature vectors were made of. When temporal signals is written, no feature is extracted: the feature vectors are made from the normalized signal values. For instance, in table 6.3, the selected features are

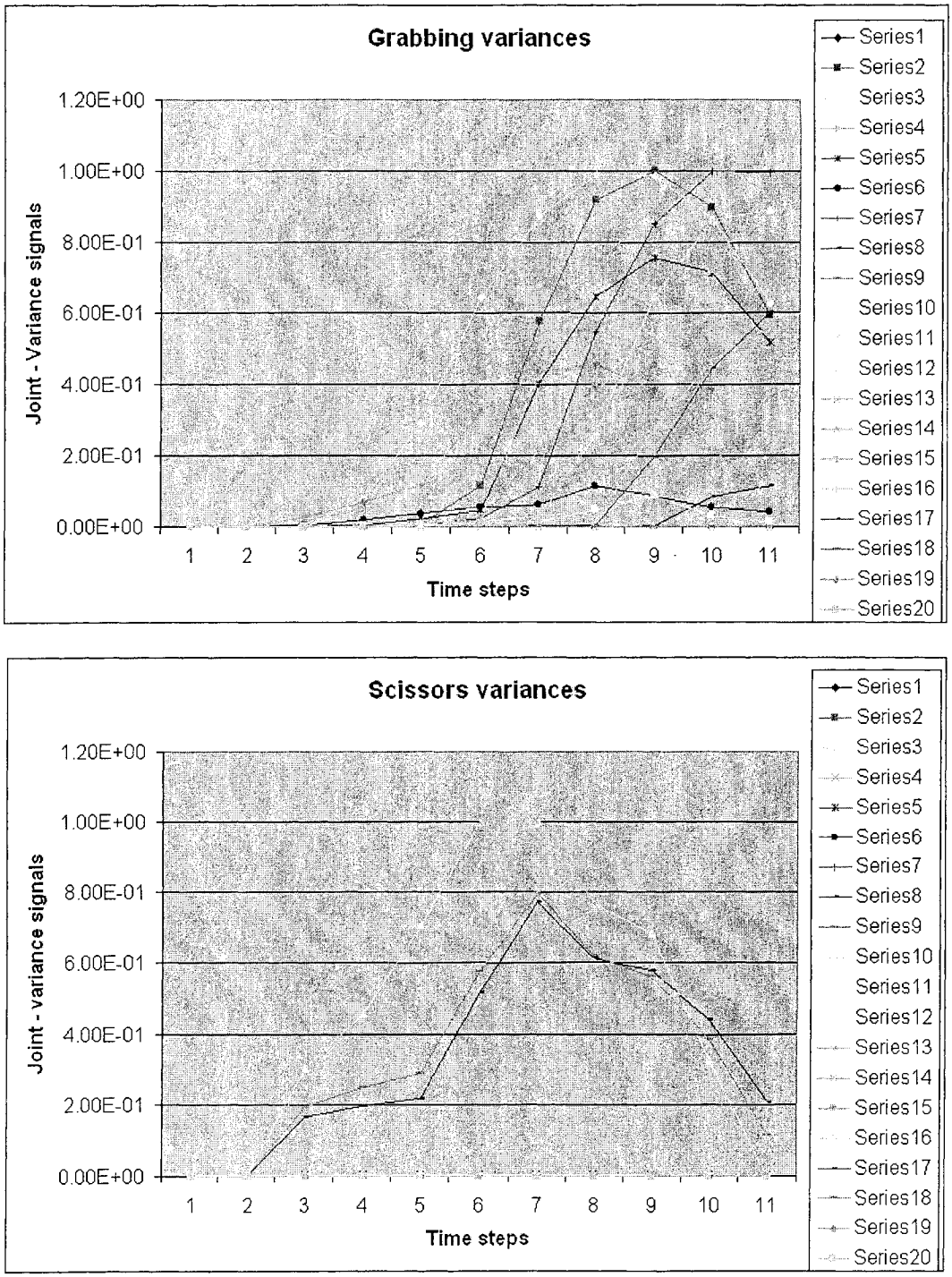

Figure 6.2: *Padding* effect on *variance* feature

mean and *variance*. Last, each table indicates the size of the codebook chosen for each method (LBG and k-mean) under brackets. The codebooks extracted for table 6.3 have respectively a size of 8 and 6 codewords.

Typically, the number of different feature vectors generated by the gesture set C is between 630 and 760 depending on the features chosen. So our following vector quantization tests with codebook sizes ranging from 6 to 32, represent significant reduction ratios (more than 95%).

No feature extraction: temporal signals only

Gesture model	LBG	k-mean
C sign	2,2,2,2,2,2,2,2,2,2	5,5,5,5,5,5,5,5,5,5
fist	0,0,0,0,0,0,0,0,0,0	4,4,4,4,4,4,4,4,4,4
flat palm	7,7,7,7,7,7,7,7,7,7	0,0,0,0,0,0,0,0,0,0
grabbing	7,7,3,6,6,6,2,4,0,0,0	0,0,0,5,5,5,5,5,4,4,4
gun	1,1,1,1,1,1,1,1,1,1,1	1,1,1,1,1,1,1,1,1,1,1
scissors	5,1,1,5,5,5,5,5,5,5,5	2,2,2,2,2,2,2,2,2,2,2
victory	5,5,5,5,5,5,5,5,5,5,5	2,2,2,2,2,2,2,2,2,2,2

Table 6.1: Temporal signals, codebook sizes [8,6]

On table 6.1 displaying the gesture model encodings when specifying a codebook size of 6, regarding the LBG method, we can see that every sign is well encoded, that the *grabbing* gesture's encoding is remarkably nice looking: starting from a *flat palm* configuration, going through transitory hand shapes including the *C sign* one and finishing in a *fist* shape. Nevertheless the encoding of the *scissors* gesture is mostly confounded with the *victory* posture.

The *k-mean* method results, although coarser than the one of the LBG, are simlar in their achievements: *grabbing* is approximated as *flat palm*→*C sign*→*fist* and *scissors* entirely mistaken with *victory*.

As we can see on table 6.2, reducing the concision degrades the quality of the encodings for the LBG method: the *C sign* occurence in the *grabbing* encoding disappeared, *scissors* and *victory* are indissociable. On the contrary, the *k-mean* method gains in encoding precision and we even discern a periodic pattern in *scissors*.

Gesture model	LBG	k-mean
C sign	14,14,14,14,14,14,14,14,14,14,14	03,03,03,03,03,03,03,03,03,03,03
fist	08,08,08,08,08,08,08,08,08,08,08	05,05,05,05,05,05,05,05,05,05,05
flat palm	23,23,23,23,23,23,23,23,23,23,23	18,18,18,18,18,18,18,18,18,18,18
grabbing	23,11,19,3,28,28,18,20,12,08,08	18,18,02,04,04,19,19,13,13,05,07
gun	09,09,09,09,09,09,09,09,09,09,09	12,12,12,12,12,12,12,12,12,12,12
scissors	21,21,21,21,21,21,21,21,21,21,21	16,08,08,16,16,01,01,16,08,08,16
victory	21,21,21,21,21,21,21,21,21,21,21	00,00,00,00,00,00,00,00,00,00,00

Table 6.2: Temporal signals, codebook sizes [32,20]

Temporal features: *mean,variance*

Gesture model	LBG	k-mean
C sign	2,2,2,2,2,2,2,2,2,2	1,1,1,1,1,1,1,1,1,1
fist	0,0,0,0,0,0,0,0,0,0	5,5,5,5,5,5,5,5,5,5
flat palm	7,7,7,7,7,7,7,7,7,7	3,3,3,3,3,3,3,3,3,3
grabbing	7,7,7,7,3,3,3,3,3,6	3,3,3,3,3,3,2,2,2,2
gun	1,1,1,1,1,1,1,1,1,1	0,0,0,0,0,0,0,0,0,0
scissors	5,5,5,5,5,5,5,5,5,5	0,0,0,0,0,0,0,0,0,0
victory	5,5,5,5,5,5,5,5,5,5	0,0,0,0,0,0,0,0,0,0

Table 6.3: *Mean,variance* features, codebook sizes [8,6]

When specifying a size of 6 for the codebook, the encodings obtained with both codebook building methods when using temporal features are pale in comparison to their previous counterparts. Indeed, table 6.3 shows that the temporal behavior of dynamic gestures is less visible and gesture amalgam is more obvious: with the *k-mean* method all last three gestures are approximated by a continuous 0 encoding.

Augmenting the precision by encreasing the codebook size does slightly improve the situation (table 6.4) but not to the point of achieving the clarity of encodings based on temporal signals. We suggest that the TFP is at the source of this loss of clarity for the variances of all types of gestures are around zero during the first time steps. Amusingly, one may point out that our incentive to

Gesture model	LBG	k-mean
C sign	10,10,10,10,10,10,10,10,10,10	05,05,05,05,05,05,05,05,05,05
fist	08,08,08,08,08,08,08,08,08,08	13,13,13,13,13,13,13,13,13,13
flat palm	23,23,23,23,23,23,23,23,23,23	11,11,11,11,11,11,11,11,11,11
grabbing	23,23,23,15,15,7,11,11,27,14,14	01,01,01,01,16,14,14,14,02,02,02
gun	09,09,09,09,09,09,09,09,09,09	06,06,06,06,06,06,06,06,06,06
scissors	13,05,05,05,05,05,05,21,21,05	15,00,00,00,00,00,18,18,18,18,00
victory	13,13,13,13,13,13,13,13,13,13	17,17,17,17,17,17,17,17,17,17

Table 6.4: *Mean,variance* features, codebook sizes [32,20]

capture gesture dynamics through feature extraction, hampers the dynamics' visibility in the encodings. Moreover, using both temporal features is counter-productive, since we deal with feature vectors with twice the size of temporal vectors (2*20 features versus 20 joint measurements).

Static features: finger states and abductions

Gesture model	LBG	k-mean
C sign	6,6,6,6,6,6,6,6,6,6	0,0,0,0,0,0,0,0,0,0
fist	2,2,2,2,2,2,2,2,2,2	0,0,0,0,0,0,0,0,0,0
flat palm	3,3,3,3,3,3,3,3,3,3	5,5,5,5,5,5,5,5,5,5
grabbing	3,3,3,4,4,4,4,2,2,2,2	5,5,5,2,2,2,2,0,0,0,0
gun	1,1,1,1,1,1,1,1,1,1	1,1,1,1,1,1,1,1,1,1
scissors	5,1,1,5,5,1,5,5,1,1,5	4,1,1,4,4,1,1,4,1,1,4
victory	5,5,5,5,5,5,5,5,5,5	4,4,4,4,4,4,4,4,4,4

Table 6.5: *Finger states,abduction* features, codebook sizes [8,6]

After the disappointment from the previous feature choice, table 6.5 brings us some heartwarming news. The encoding resulting from the LBG method may be hold as ideal for the evolutions of *grabbing* and *scissors* are particular well captured (especially the periodic trait of the latter). The *k-mean* side is as bright, with a small cloud being the lack of distinction between *C sign* and *fist*. As these two signs are quite similar, their feature vector representations

are forming an unique cluster, which can be broken done when increasing the size of the codebook to 10.

Gesture model	LBG	k-mean
C sign	10,10,10,10,10,10,10,10,10,10	02,02,02,02,02,02,02,02,02,02
fist	22,22,22,22,22,22,22,22,22,22	15,15,15,15,15,15,15,15,15,15
flat palm	23,23,23,23,23,23,23,23,23,23	05,05,05,05,05,05,05,05,05,05
grabbing	19,19,11,03,20,20,12,02,18,18,10	05,05,05,17,17,19,19,14,14,14,14
gun	17,17,17,17,17,17,17,17,17,17	13,13,13,13,13,13,13,13,13,13
scissors	21,24,24,24,24,24,24,21,24,24,21	12,04,04,12,12,04,12,16,12,12,12
victory	13,13,13,13,13,13,13,13,13,13	01,01,01,01,01,01,01,01,01,01

Table 6.6: *Finger states,abduction* features, codebook sizes [32,20]

Table 6.6 allows us to witness the lack of concision when augmenting the codebook size beyond its optimality: although the *k-mean* results reflect now the difference between *C sign* and *fist*, we lost the enviable view of *scissors* as an alternation between *victory* and *gun*.

Eventually, this combination of features proves to be a good choice since it generates desirable gesture encodings and also reduces the number of values to examine (5+5). Since the *scissors* gesture relies on capturing the dynamics of a unique abduction joint, leaving aside the *abduction* feature makes the discrimination between the last three gestures more difficult.

All previous features

Does more means better?

Table 6.7 displaying the methods' results for a limited codebook size, presents the previously acquainted indiscrimination issues between *scissors* and *victory* for LBG, *C sign* and *fist* for the *k-mean* method. To that extent, the shortcoming of both feature groups were cumulated.

As the size of the codebook gets greater (table 6.8), the encoding precision becomes better. Yet, it does not have the concision achieved with only the *state finger* and *abduction* features and is more computationnally intensive than the raw temporal signal solution: the feature vector dimension is 50 (20+20+5+5=50).

Gesture model	LBG	k-mean
C sign	2,2,2,2,2,2,2,2,2,2	2,2,2,2,2,2,2,2,2,2
fist	0,0,0,0,0,0,0,0,0,0	2,2,2,2,2,2,2,2,2,2
flat palm	7,7,7,7,7,7,7,7,7,7	0,0,0,0,0,0,0,0,0,0
grabbing	7,7,7,3,3,3,3,6,6,6	0,0,0,0,0,1,1,1,1,1
gun	1,1,1,1,1,1,1,1,1,1	5,5,5,5,5,5,5,5,5,5
scissors	5,5,5,5,5,5,5,5,1,1,5	4,3,3,3,3,3,3,3,3,3
victory	5,5,5,5,5,5,5,5,5,5	4,4,4,4,4,4,4,4,4,4

Table 6.7: *Mean,variance,finger states,abduction* features, codebook sizes [8,6]

Gesture model	LBG	k-mean
C sign	10,10,10,10,10,10,10,10,10,10	03,03,03,03,03,03,03,03,03,03
fist	16,16,16,16,16,16,16,16,16,16	18,18,18,18,18,18,18,18,18,18
flat palm	23,23,23,23,23,23,23,23,23,23	11,11,11,11,11,11,11,11,11,11
grabbing	23,15,11,11,19,03,03,14,30,14,14	07,07,02,12,12,00,08,08,06,06,06
gun	09,09,09,09,09,09,09,09,09,09	01,01,01,01,01,01,01,01,01,01
scissors	13,05,05,21,25,05,21,21,21,05,21	05,16,16,16,16,16,19,19,19,19,16
victory	13,13,13,13,13,13,13,13,13,13	05,05,05,05,05,05,05,05,05,05

Table 6.8: *Mean,variance,finger states,abduction* features, codebook sizes [32,20]

6.2.3 Conclusion

This quick overview about the feature (non-)selection' influence on the gesture encoding profiles lead us to conclude that the TFP made temporal feature irrelevant. As we ought to discard them, we retain only the temporal joint signals or the group of static feature to test the recognition process.

For the sake of being synthetic, we only showed the effects of the vector quantization on gesture models. The reality when considering samples encodings is more intricate: usually not all samples of a gesture have similar encodings. Posture samples may use different codeword sequences and sometimes are *broken*: using more than one codeword for their encoding. With dynamic gestures, samples encodings reflect closely the differences in execution speed between recording sessions.

The heterogeneity between samples encodings may be used as an indicator about the classification difficulty and underlines how primitive is our averaging strategy to obtain gesture models.

6.3 Classification

6.3.1 Evaluation methods

To evaluate the performance of the GR process, we set some gesture samples as inputs. We thought of three methods to select the test samples:

- The **substitution** method (S): the training and testing sets are identical. Using this method is relatively light; we train the classifier once and then perform the testing in one pass. Obviously this method is optimistically biased, since the tested inputs are already known by the classifier. In order to obtain some more objective classification estimations, we introduced the next methods.

- The **half-out** method (H): we split the set of available samples in two using one half for the training, sparing the other for the testing. As the classifier is not familiar with the testing samples, this method is not skewed. However, the method requires training sets of big dimensions, it does not make a very efficient use of the data for the training.

- The **one-out-turning** method (O): the whole set of samples is used for training with exception of one sample spared for the testing. Once the test has been perfornmed, we repeat the previous operations by excluding another test sample. The process is repeated as long as their are samples that have not been tested. The method is not skewed and uses the data set intensively. Nevertheless, this method may be too expensive with respect to classifiers such as neural networks because of their training may prove to be too time consumming.

In order to interpret the results, we pay attention to two probabilities:

- The **missclassification** probability (P_{nd}): chances that a gesture be detected as another one.

- The **non-detection** probability (P_{miss}): chances that a gesture triggers no detection event at all.

The probability of sucessful classification of a gesture can be deduced from the two previous ones (its value being 1-(P_{nd}+ P_{miss})).

Considering the limited size of our gesture sets (only 10 samples for each gesture), the H and S probabilitic results are not reliable: a single recognition failure for one sample will hinder the recognition rate by at least 10 %. Experimentally, we used the S method to see roughly where the problems were located and tweak the framework's parameters accordingly. Moreover we used it with the neural network classifier for this latter's training would make the O method too time expensive. On the contrary, the classification results for the template matching or HMM approaches were obtained with the O method.

The results displayed in the following tables were obtained by using the *one-out-turning* method for the HMM and template matching classifiers whereas the neural network estimates were calculated by the *substitution* method (only one training required and data set completely used).

Since the classification stage takes on after the representation step, these results reflect not only classifier settings but also the decision made to represent gestures.

6.3.2 Template matching approach

As a preliminary step, we gladly noticed the equivalence of using thresholds based on the metric values d or the similarities ($\frac{1}{1+d}$). For every metric chosen, and every threshold strategy (mean, mean plus variance, maximum), the classification error probabilities are the same.

- *Euclidian* metric:

  When evaluating the classification performances of the template matching approach on our gesture set, we modified a bit the version using the *variance* for thresholding the gesture classes. Indeed, as adding one time the variance value did not appear sufficient to modify the results obtained with the *mean* thresholds, we added it 3 times.

  Table 6.9, in which are displayed the error probabilities [P_nd,P_miss], conforms the intuition: as the classes radii increase, the non detection problem is turned progressively into a missclassification one.

Gestures	Mean	Mean + 3*Variance	Max
C sign	[38;0]	[38;0]	[0;0]
fist	[56;0]	[47;0]	[0;0]
flat palm	[24;0]	[22;0]	[0;0]
grabbing	[66;0]	[57;0]	[0;0]
gun	[20;0]	[20;0]	[0;0]
scissors	[58;0]	[17;24]	[0;19]
victory	[18;0]	[8;20]	[0;4]

Table 6.9: Without vector quantization, euclidian classification

Gestures	Mean	Mean + 3*Variance	Max
C sign	[0;10]	[0;10]	[0;10]
fist	[0;20]	[0;20]	[0;20]
flat palm	[0;0]	[0;0]	[0;0]
grabbing	[0;0]	[0;0]	[0;0]
gun	[0;0]	[0;0]	[0;0]
scissors	[0;20]	[0;20]	[0;20]
victory	[0;0]	[0;0]	[0;0]

Table 6.10: Without vector quantization, euclidian classification

Observing the table 6.10, in which we display the error probabilities when using some vector quantization beforehand, we might be surprised by the identity of the error results, even when the gesture classes radii have change significantly. However, this can be explained by the fact that we do no longer take the distance between feature vectors but between their fixed representations (codewords).

- *DTW* and *Mahalanobis* metrics:

Experimentally, we observed that both Mahalanobis and DTW metrics behaved in a similar manner than their euclidian counterpart, the best results being achieved with the *max* thresholding option. The reason of this is that gesture classes are more finely enclosing their samples, therefore

limiting missclassification effects.

6.3.3 Neural network approach

Using the neural network strategy to classify recorded samples proves very efficient: indeed with sufficient training we achieved recognition rate above 95% . When errors occur, they are always caused by some missclassification. We have only worked with temporal signals and feature vector representations, for the input layer format was not designed to cope with vector quantization encodings.

6.3.4 Hmm Viterbi approach

In the table 6.11, we only mentioned 2 gestures, namely *C sign* and *fist*, since for the rest of the *C* test set, there was no classification error. Moreover the classification percentages can be easily understood when looking at the samples encodings:

- The*C sign*: all samples are encoded as a sequene of the **6**-th codeword except the first one, which uses the **2**-nd codeword.
- The *fist* posture: all samples are sequences of the **2**-nd codeword except 2 of them, sequences of the **6**-th.

The missclassification error origin is clear, possibly originating from a lack of carefulness when recording both gestures.

Gestures	Non detection	Missclassification
C sign	0	10
fist	0	20

Table 6.11: Open Viterbi approach classification results

6.4 Real world evaluation

The previous classification evaluation gives only a limited idea about the framework's behavior in real time. This limitation comes from the fact that classification only tests the recognition process on meaningful (gesture samples) and

well defined (samples carefully recorded) inputs, whereas the real time captured data are more diverse and subject to noise.

To gauge the efficiency of the framework for real time data, we decided to record gestural inputs over a definite period of time during which the user would execute a predefinite sequence of gestures. The log file we used contained the following sequences of gestures:

- Log = {*fist,flat palm,victory,grabbing,C sign,grabbing*} = {1,2,6,3,0,3}

The numeric sequence is the translation of the gestures into their indices. The totality of the log file covers about 1 minute and was recorded with the same capture parameter as the gesture samples, namely with a sampling rate of **11.75** Hz.

In a second time, log files are loaded into our framework and set as inputs. Once all recognition parameters have been decided, we launch the online recognition process and save the results of the different stages in files. This way of proceeding, based on identical gestural input sequences, allows us to see more objectively the recognition results than a user qualitative appreciation.

- *Viterbi approach:*
  On figure 6.3, a favorable case , we can observe the sequence {1,0,6,3,1,3}, which almost matches the theoretical sequence. The problems are caused by missclassification between the pairs (*C sign*, *flat palm*) and (*C sign*, *Fist*), which could be foreseen from the classification results in table 6.11. We tried to improve the discrimation between these similar postures by increasing the size of the codebook but without any conclusive results: the detection was messier, with even more missclassification problems.

  In order to see how the recognition behaved in front of 'noisy' gestural inputs, we considered the same log but restricting the gestures to the test set *A* and then *B*. With *A*, we simply detected nothing, not even any occurence of *victory* that seemed to be quite noticeable (only gesture detected in other less advantageous configurations). With *B*, all three signs *fist*,*C sign* and *flat palm* are detected as *C sign*...

- *Template matching approach:*
  Since the template matching classification results were pretty good looking, we tried this approach on the log file. The results were really desappointing with all 3 metrics: we merely achieved to detect the *victory*

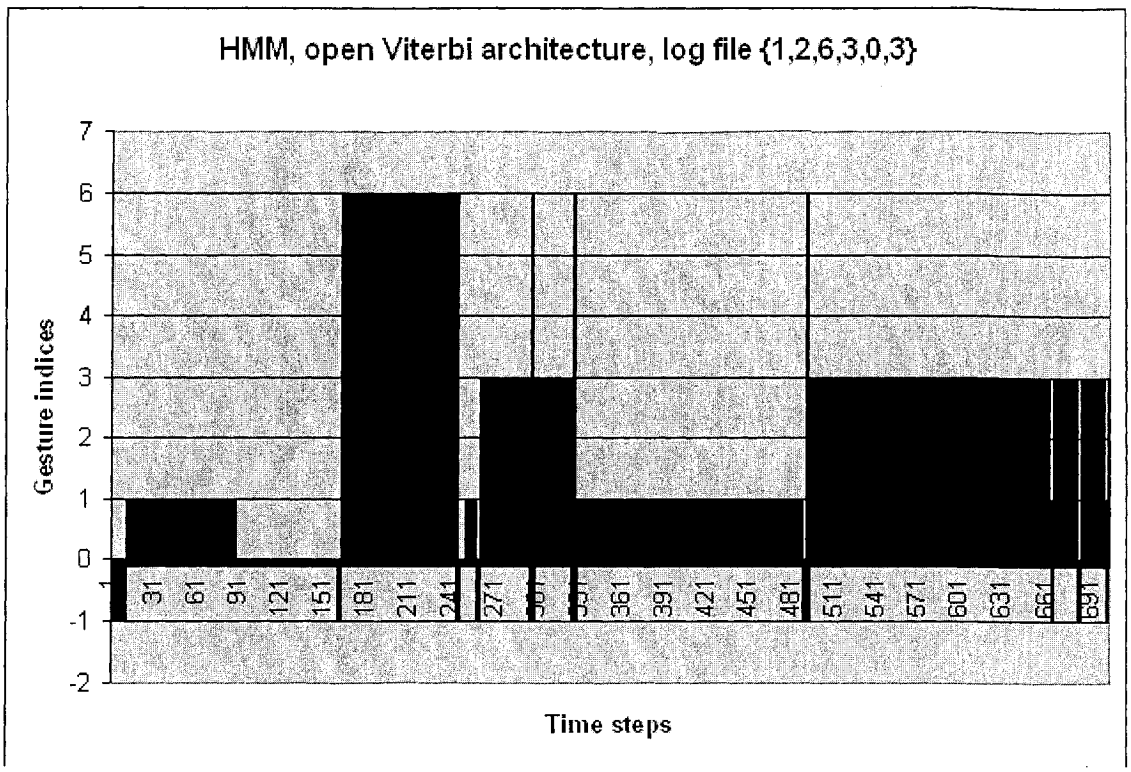

Figure 6.3: HMM, open Viterbi architecture, applied on log file

gesture. This blatant failure must be caused by the very limited number of gesture samples we had: even taking the *maximum* option for setting the thresholds did not bring any detection. As the natural ways of performing a gesture in real time are too diverse to be captured by a static strategy such as template matching, we renounced to this later.

Considering the classification results of the closed HMM approach (with a final state linked back to the initial state), we tried it without much hope on the log file. Indeed, there was no detection at all and the explanation lies in that the most probable state sequences given by the viterbi algorithm are composed of the repetition of the final and initial states (no use of the gesture branchs). Indeed, we set the observation probabilities of these two states to the average of the observation probabilities of the final

(respectively initial) states of the gesture branchs. As a result, every type of observations is quite probable in both states.

- *Neural network approach:*
  The neural network approach proves to be robust at detecting most gestures with comparable results as the Viterbi approach. However, it has much less flexibility (training time) and the user does not have the impression that dynamics of the hand are captured. For instance, when performing *victory* or *gun*, changing moderately the configuration of some fingers, we will see the *scissors* gesture detected (although the hand is poised). Compared to the Viterbi approach, it has an advantage if we hold missclassification errors more benign than non detection ones : when testing the log file with the tests sets A and B, the gestures excluded were interpreted as gestures of the subset (typically with the A set, *grabbing* was converted into *gun*).

Chapter 7

Conclusion

When engaging this domain, as freshcomers in the pattern recognition area, we thought that the challenge would be solved in two steps: first *retrieving* the measurements then *deciding* what gestures they express. For the decision taking process, fuzzy logic seemed appropriate: mutually excluding hard coded rules described each *gesture*. At that time, our gestures were simple signs.

This very first approach worked for a very limited number of signs, was unsatifying with respect to two elements:

- First, everything from the gesture set to the description rules was hard-coded. This was not admissible if we wanted to obtain a flexible recognition framework. Thus, a necessary improvement was to find a way for constructing description rules for arbitrary signs.

- Second, when we thought about extending the recognition to dynamic gestures, the description rules complexities dramatically increased since they should take the inputs of several time steps into consideration.

Weighing these two issues, we judged that fuzzy logic was not suitable for our problem and turned ourself to neural networks. On the contrary to the previous approach, classification with neural networks is an implicit process. We give them gestures as inputs and look at their outputs for probabilities of gestures being executed. In that they satisfied our automatization expectation but using a neural network is quite time consuming during its training part. Indeed, with a relatively small number of gestures (7), it can take more than

15 minutes and much parameter (learning rate, momentum) tweaking to obtain a good classification rate. Moreover, this classification strategy is not flexible with respect to gesture sets: adding or removing a gesture entails an enlistment for another training session. Moreover, when experimenting, we never had the impression of capturing the dynamics of a gesture: it was still possible to get dynamic gestures recognized by holding one's hand poised.

The previous observation triggered a need for capturing more explicitely hand shapes variations, which resulted in first considering a supplementary stage before performing the classification (*representation*) and a more explicit handling of classification through template matching. The metrics for matching gesture models with inputs were chosen with different intentions:

- The euclidian metric was retained for its simplicity, to get acquainted quickly with the template matching strategies characteristics.
- We were interested in testing the Mahalanobis distance for, through a statistical analysis, gestures classes would be more precisely characterized, possibly reducing the missclassification issues of the euclidian metric.
- Kheogh [30], by his time series classification results, was a source of inspiration for using the dynamic time warping approach to classify our signals subject to shortening and stretching.

In parallel, we tried to improve the temporal hand behavior analysis by extracting features and subsequently using vector quantization. We only considered intuitive features, for otherwise we would have been drawn too deep in the digital signal processing domain. The vector quantization step was added, for it was necessary for our last classification trial and eventually proved to be of great help in evaluating gesture representations. Indeed, observing resultant gesture encodings indicates how well the temporal evolution is rendered, which gestures might be subject to missclassification. These observations are very useful for selecting features appropriate for a given gesture set.

As we observed the different gesture encodings, the idea to 'match' them to detect gestures came to our made and we engaged in template matching. However, when it came to real time recognition, it disappointed us completely.

Finally, we were keen on trying an Viterbi HMM approach for the following reasons:

- The design tightly reflects gestures temporal behaviors since it is based on the vector quantization results.
- The *left-right* architecture with self transition loop theoretically makes the model able to cope gracefully with execution speed variations.
- Adding or removing gestures only requires minimal modification of the architecture (other gesture branches remain identical).

7.1 Future work

From the different experiments we performed, if we were to keep investigating this issue further, we would concentrate on:

- Exploiting finger signals correlations to reduce the number of signals to examine.
- Refining the vector quantization and exploring alternatives to define gesture models. For instance, instead of *averaging* sample encodings, we could look at common subsequences.
- Experimenting further with the Viterbi architecture, by training the gesture branchs individually with the Baum-Welch algorithm.
- Currently, the GUI of the Gesturon GR interface has been written with the GLUI graphical toolkit. This latter creates a process for managing the different graphical components in different threads, which prevents any direct use of the Gesturon interface from an existing multithreaded application. A first, easily achievable refinement, will consist of rewriting the GUI part in Java, linking it with the core of the recognition using the JNI. Once this modification performed, we will be able to compile the whole GR interface in some dll usable by an C++ or Java applications.

Gesture recognition is a challenging field. We tried to tackle this issue with much determination but at last we must acknowledge that obtaining reliable recognition results useful for interfaces, as presented in the *Minority Report*, is still in the domain of science fiction.

Bibliography

[1] http://www.immersion.com/3d/products/cyber_glove.php

[2] http://www.ascension-tech.com/products/flockofbirds.php

[3] http://www.lornet.org/i2lor/index.htm

[4] http://java.sun.com/products/java-media/3D/

[5] http://www.j3d.org/

[6] http://www.rolandit.com/games/Peripherals/viewperipheral.asp?GID=7

[7] http://java.sun.com/docs/books/tutorial/native1.1/

[8] http://www.nigels.com/glt/glui/

[9] http://www.data-compression.com/vq.shtml

[10] http://www.vrealities.com/glove.html

[11] http://www.washington.edu/newsroom/design.html

[12] T. Métais, N.D. Georganas, *A Glove Gesture Interface*, in proceedings of the Biennial Symposium. on Communications, Kingston, 2004.

[13] T.O. Ellis, J.F. Heafner, W.L. Sibley, *The "GRAIL" Project*, in the 8th Annual IEEE Symposium on Human Factors in Electronics 1967.

[14] K. Morrison, S. McKenna, *Contact-Free Recognition of User-Defined Gestures as a Means of Computer Access for the Physically Disabled*, in proceedings of CWUAAT '02, 2002.

[15] R.M. Krauss, Y. Chen, R. Gottesman, *Lexical gestures and lexical access: A process model*, McNeill, in Language and Gesture. pp.261-284. Cambridge: CUP, 2000.

[16] G.J. Miao, M.A. Clements, *Digital Signal Processing and Statistical Classification*, Artech House.

[17] S. Theodoridis, K. Koutroumbas, *Pattern Recognition, second edition*, Elsevier academic press, ISBN 0-12-685875-6

[18] E. Micheli-Tzanakou, *Supervised and Unsupervised Pattern Recognition, Feature Extraction and Computational Intelligence*, Industrial Electronics Series, CRC Press, ISBN 0-8493-2278-2.

[19] R.P. Ramachandran, R. Mammone, *Modern Methods Of Speech Processing*, Kluwer Academic Publishers, ISBN 0-7923-9607-3

[20] X.D. Huang,Y. Ariki, M.A. Jack, *Hidden Markov Models for Speech Recognition*, Edinburgh University Press, ISBN 0-7486-0162-7

[21] M. Allerhand, *Knowledge-Based Speech Pattern Recognition*, Kogan Page, ISBN 1-85091-260-2

[22] Y.Y. Zhu, *High Performance Data Mining in Time Series: Techniques and Case Studies*, Ph.D dissertation, New York University, 2004.

[23] J.Jr. LaViola, *Whole-Hand and Speech Input in Virtual Environments*, Master of Science thesis, Brown University, 1999.

[24] D.J. Sturman , *Whole-hand Input*, Ph.D dissertation, Massachusetts Institute of Technology, 1992.

[25] A. Sandberg, *Gesture Recognition using Neural Networks*, Master of Science thesis, Stockholm University, 1997.

[26] D.O. Tanguay, *Hidden Markov Models for Gesture Recognition*, Master of Engineering thesis, Massachusetts Institute of Technology, 1995.

[27] P.A. Harling, *Gesture Input using Neural Networks*, report, University of York, 1993

[28] J.b Eisenstein, S. Ghandeharizadeh, L. Golubchik, C. Shahabi, D. Yan, R. Zimmermann, *Device Independence and Extensibility in Gesture Recognition*, Department of Computer Science, at the Virtual Reality Conference, Los Angeles, 2003.

[29] R. Watson, *A Survey of Gesture Recognition Techniques*, technical report TCD-CS-93-11, Department of Computer Science, Trinity College Dublin, 1993.

[30] S. Chu, E. Keogh,D. Hart,M. Pazzani, *Iterative Deepening Dynamic Time Warping for Time Series*, in proceedings of the 2002 IEEE International Conference on Data Mining, Japan, 2002.

[31] P.J. Werbos, *Backpropagation through time: what it does and how to do it.*, in the Proceedings of IEEE, 1990.

[32] K. Murakami, H. Taguchi. Gesture recognition using recurrent neural networks. In Proceedings of the Conference on Human Factors and Computing Systems (CHI '91), Louisiana, 1991.

[33] P. Vamplew, A. Adams, *Recognition and anticipation of hand motions using a recurrent neural network*, at IEEE International Conference on Neural Networks, Perth, 1995

[34] S. Fels, G. Hinton, *Glove-Talk: A Neural Network Interface Between a Data-Glove and a Speech Synthesiser*, in IEEE Transactions on Neural Networks, 1993.

[35] K. Boehm, W. Broll, M. Sokolewicz, *Dynamic Gesture Recognition Using Neural Networks; A Fundament for Advanced Interaction Construction*, Proceedings of the SPIE, 1994.

[36] Y. Linde, A. Buzo, R.M. Gray, *An Algorithm for Vector Quantizer Design*, IEEE Transactions on Communications, 1980.

[37] J. Yang, Y. Xu, *Hidden Markov Model for Gesture Recognition*, technical report CMU-RI-TR-94 10, Carnegie Mellow University, 1994.

[38] N. Liu, B.C. Lovell, P.J. kootsookos, R.I.A. Davis, *Model Structure Selection & Training Algorithms for a HMM Gesture Recognition System*, in

the Ninth International Workshop on Frontiers in Handwriting Recognition (IWFHR'04), Japan, 2004.

[39] T. Starner, A. Pentland, *Visual Recognition of American Sign Language Using Hidden Markov Models*, Proceedings of the International Workshop on Automated Face and Gesture Recognition, Zurich, 1995.

[40] L.R. Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, Proceedings of the IEEE, vol. 77, no. 2, pp. 257 - 286, 1989.