UNIVERSITÀ DI PISA

College of Engineering

Master's Degree in Automation Engineering

DESIGN AND DEVELOPMENT OF A HUMAN GESTURE RECOGNITION SYSTEM IN TRIDIMENSIONAL INTERACTIVE VIRTUAL ENVIRONMENT.

Candidate:

Vittorio Lippi __________

Supervisors:

Prof. Carlo Alberto Avizzano ______________

Prof. Giuseppe Lipari ______________

Prof. Emanuele Ruffaldi ______________

Graduation Session of 28/02/2008
accademic Year 2006/2007
Consultation Allowed

To my parents.

Abstract

This thesis describes the design and the development of a recognition system for human gestures. The main goal of this work is to demonstrate the possibility to extract enough information, both semantic and quantitative, from the human action, to perform complex tasks in a virtual environment. To manage the complexity and the variability adaptive systems are exploited, both in building a codebook (by unsupervised neural networks), and to recognize the sequence of symbols describing a gesture (by Hidden Markov models). An high level description language for the recognized processes has been produced. It describes the environment using fuzzy logic and produces prior probabilities for recognized gestures derived from the context they are performed in.

Acknowledgments

Thanks to the professors, the researchers and the students who work at PERCRO laboratory where I had done this work, in particular to my supervisors, Carlo Alberto Avizzano and Emanuele Ruffaldi, and to Paolo Tripicchio, Edoardo Sotgiu, Otniel Pontiello, Walter Aprile, Silvia Pabon, Glauco Ferrari and Alessandro Nicoletti who provided information about hardware and software resources available at the laboratory. In particular Davide Sechi worked with me to integrate my work in a virtual environment and Massimo Sattler helped me in preparing movies of my experiments.

Thanks to Alessio Micheli, professor of the faculty of Computer Science of university of Pisa, who reviewed an italian version of chapter 5 giving me precious advices how to improve it.

Thanks to my parents, Giuseppe and Donatella who have been extremely supportive during the realization of this work, and had a substantial influence on my choice to became an engineer. Choice that gave me the opportunity to find, being in contact with the world of science and technology, a nowadays uncommon trust in human capabilities and an enthusiasm for the future.

Thanks to Silvia Bonotti who helped me to write this thesis in English by reviewing the chapters 5 and 6.

I would like to thank also some friends who helped me in several ways during the realization of this thesis and the whole university: Giacomo Ceccarelli, physicist, who has always been ready to discuss mathematical issues (and even more) with me; Nicola ”Talos” Gazzeri, colleague computer engineer, who shared with me the secrets of operating systems and the passion for Weissbier; My cousin and friend, Giovanni Vittorio ”Penna” Spina, colleague computer engineer, who has been a supplier of materials, ideas and hints about informatics and engineering issues.

Contents

List of Figures

Chapter 1

Introduction

Gesture recognition is becoming an important task in technology and science. At first it can be considered an advanced way to interface the user to an informatics system, exploiting the significance of natural human actions as commands. This would produce a greater efficiency both in using and in learning how to use the system. In [WB] the main advantages in using gesture recognition have been summarized:

- *natural interaction:* Gestures are a natural form of interaction and easy to use[1];
- *terse and powerful:* A single gesture can be used to specify both a command and its parameters;
- *direct interaction:* The hand as input device eliminates the need for intermediate transducer.

Beyond this, a system capable of reading a significance in human behavior can be useful when the task performed by the user is to be studied. Performing an operation in a virtual environment a user can test the effectiveness of his practical choices and to train himself to work in various situation, in the meanwhile the gesture recognition system could trace the steps performed: a model of the behavior of a skilled performer could be produced, and a generic user can be aided by the system to reproduce it. This can be useful in context where staff training, and producing documentation are crucial economical issues in the working process.

[1]Again in [WB], is however noted that using gesture only interface can make the user tired of performing movement in free space and requires him/her to remember the set of gestures and their meaning. Actually, this arguments are applicable to every interface system, *isn't it difficult to remember unix commands, and harmful for hand tendons to use a keyboard?*

Distinguishing gestures is a matter of identifying their characteristic by, explicitly or implicitly, creating a model for them. An experiment of gesture recognition is also a validation test for hypothesis assumed about human movements. Hence techniques of gesture recognition can gain a great advantage from the scientific study of human behavior, but also be a fundamental component of it producing and testing models for a ”from inside” description of actions.

1.1 Gesture Semantic

This section is mainly based on [MMC05] and [WB].
A gesture may be defined as physical movement of the hands, arms, face and body with the intent to convey information or meaning. Hence gesture recognition consist in the interpretation of movement as semantically meaningful commands.

The typical human computer interaction is strongly consrained by the interface and limited to manipulation of interactive devices (keyboard, mouse, trackball etc.). This interaction results in a simplified gesture the meaning of which is defined by convention. Natural human gestures are variable not just in quantitative aspects like speed and trajectory but also in their nature. Gestures can be classified according to their function:

- *semiotic:* gestures used to communicate meaningful information;
- *ergotic:* gestures used to manipulate the physical world and to create artifacts;
- *epistemic:* gestures used to learn from the environment through tactile or haptic exploration;

Being this work focused on interacting with a virtual environment, but without any force feedback, the gestures exploited would be prevalently semiotic, carrying information for the computer. Anyway the form of the gestures used would be inspired by the functional execution of ergotic gestures, for example a gesture triggering the movement of an object in the virtual environment could replicate the act of grabbing and moving it. A further classification of semiotic gestures is possible:

- *symbolic Gestures:* these are gestures that have a single meaning (by cultural convention). The ”OK” gesture and sign language are examples of this category;
- *dietic gestures:* these are the gestures of pointing or, in general, directing the listeners attention to specific events or objects in the environment;

- *iconic gesture:* these gestures are used to convey information about size, shape or orientation of the object of discourse;
- *pantomimic gestures:* these are gestures typically used showing the use of movement of some invisible tool or object in the speaker's hand.

Gesture types can be also ordered according to their speech/gesture dependency. This is described by Kendon's Continuum:

In contrast to this rich gestural taxonomy, current interaction with computers is almost entirely free of gestures. The dominant paradigm is direct manipulation. Even the most advanced gestural interfaces typically only implement symbolic or dietic gesture recognition.

1.2 Thesis Objectives

There are several gesture recognition systems in literature[2], the main issues distinguishing this work are:

- Design oriented to scalability;
- Exploiting discriminative training for gesture models, and knowledge about the environment to enhance performance;
- Let the user act through more than an agent in the virtual world;
- Coordinate use of two hands;
- Retrieve both semantic and quantitative informations from the gestures;

1.3 Application Details

The gesture identification system described in details in this thesis is a framework implemented through C, C++, Matlab/Simulink, and XVR[3]. The principal operations performed by the system are:

- reading the user position (in the sense of hand coordinates in space and fingers configurations). A sequence of sampled position is the raw description of users dynamics;

[2]The state of art in gesture recognition is the subject of chapter 2.

[3]MATLAB and Simulink are trademarks registered by Matworks inc. XVR is a scripting language oriented to virtual reality and online application

- processing the stream of data to obtain a description of the gestures as a sequence of symbols (codebook);
- creating gesture models based on collected examples;
- real time gesture recognition;
- controllng a virtual environment according to user's actions;
- retrieve useful informations from the environment;

The hardware exploited to read users position consist in a electromagnetic sensor of position and orientation and a sensorized glove, described in details in 7. The construction of a codebook, and the real time encoding are performed with Matlab as explained in 4. Sequence recognition and a logical description of the environment are implemented by two libraries built in C++, based on the principia respectively explained in 5 and 6.

Chapter 2

State of Art

2.1 Main issues in gesture recognition

Gesture Analysis requires several stages. First of all data about the gesture performed should be in some way collected. There are several kind of sensors capable of retrieving significant data about human positions and movement: an overview about possible hardware solutions is presented in 2.2, while more details about the hardware effectively used in this work are presented in chapter 7. The data collected are then encoded in a useful way. This means that a code identifying interesting user's dynamic configuration should be somehow defined. The set of symbols used and the encoding rules to obtain them from input data are known as *codebook*. Using a codebook is a way to make the stream of input data leaner, aiding the efficiency of the next elaboration steps. In for details about codebook definition in this work see chapter 4. The last operation is the recognition of the symbols sequence produced encoding the input data. Actually this step can incorporate the former, using a classification system that takes as input the raw sampled data, but this solution is not optimal as will be clear in the next paragraphs. Sequence recognition is the core process of gesture recognition. Beside the general approach given in this chapter (paragraph 2.5) sequence recognition is described with details concerning this work in chapter 5 and paragraph A.2, the appendix B describes sequences recognition using *Hidden Markov Models*.

2.2 Capturing System

In order to classify human gestures an proper hardware is needed. Considering the complex nature of human movements the acquisition hardware should be able to retrieve a lot of data (compared to the ones required for speech recognition, or for the common interface devices). Specific solutions are hence needed. The systems

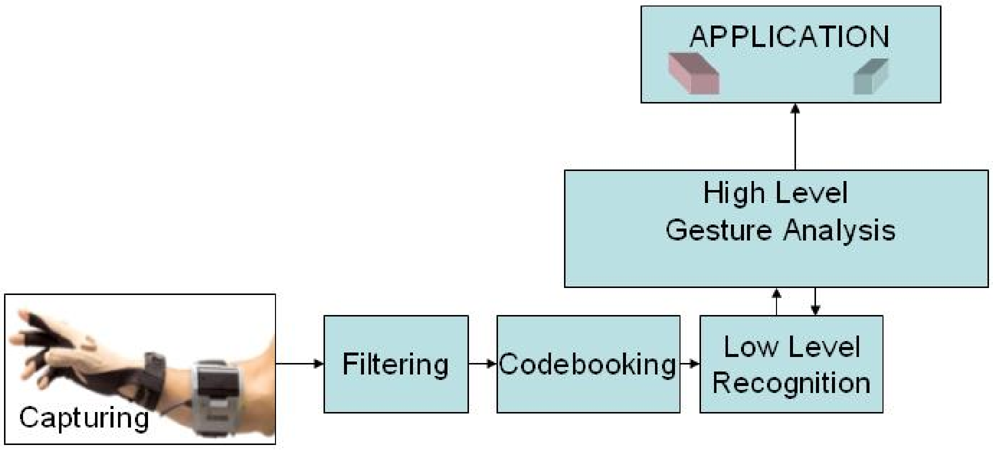


Figure 2.1: A scheme of a generic gesture recognition system

for gesture capturing consist basically in visual recognition and recognition based on weared sensors.

2.2.1 Visual Recognition

Visual recognition exploits cameras to retrieve users position. Commercial webcams are not usually suitable for this purpose. This is because the usual sampling ratio of 60 fps is too slow compared to human dinamics. The recognition is often based on markers tracking, like in the Vicon system, described in section 7.3. For hand recognition gloves of different colors can be used as visual markers [AJ05]. Identifying hand and finger position from hand shape (a so called *blob*) is the base of several systems like [MB97]

2.2.2 Haptic Interfaces

Haptic systems interfaces the user via the sense of touch by applying forces, vibrations and/or motions to the user. This mechanical stimulation may be used to assist in the creation of virtual objects (objects existing only in a computer simulation), for control of such virtual objects, and to enhance the remote control of machines and devices. To produce a feedback the aptic interface should gather interesting informations from user actions. Gesture recognition can be the most important component of an haptic application. In [JS02] a device to teach to write japanese characters is presented. A force feedback helps the user to move the pen (the haptic device) on the right trajectory. In order to decide which correction apply on user action, the system should recognize the written character using the

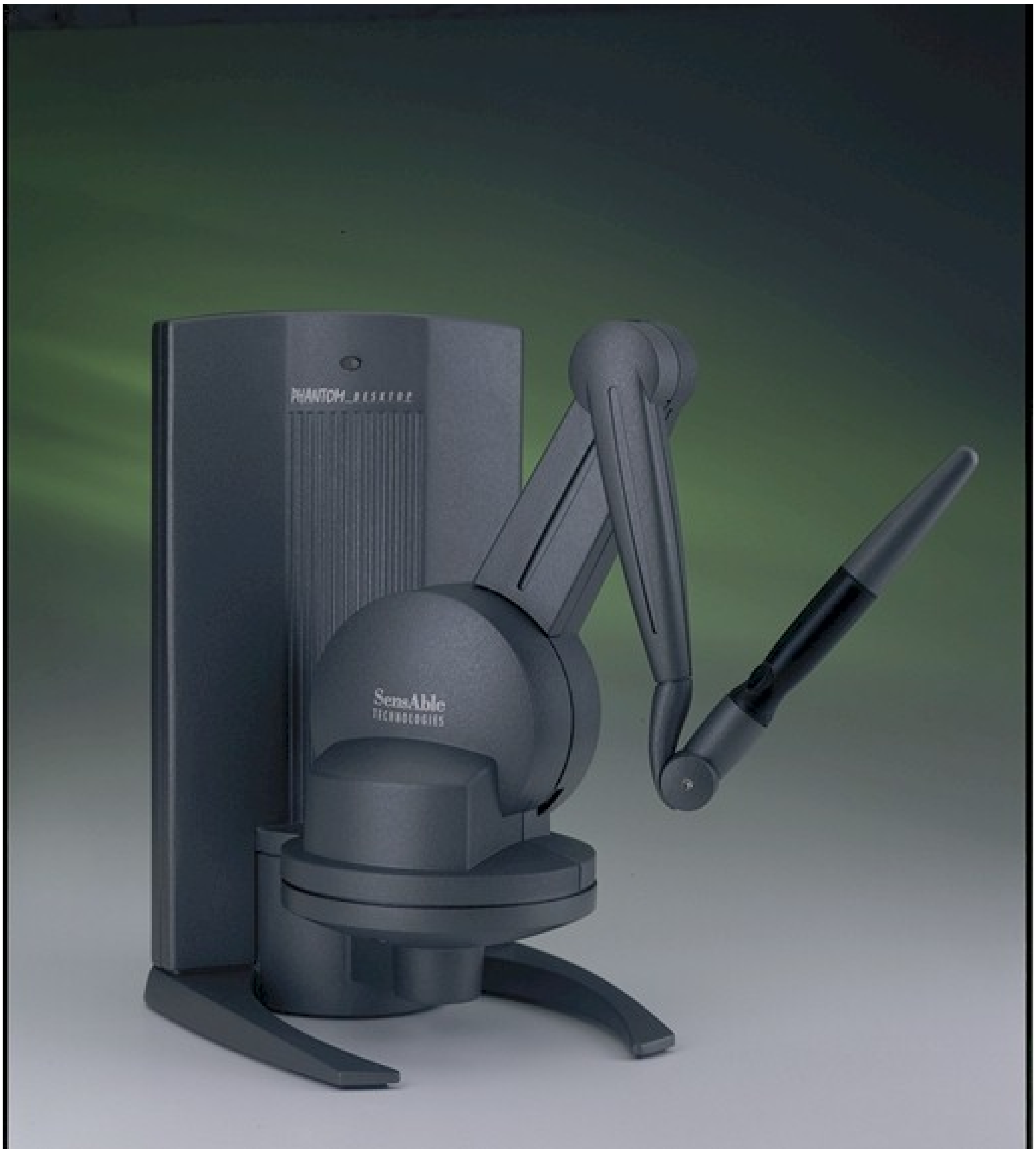


Figure 2.2: Phantom, a commercial haptic device produced by SensAble

informations from the haptic device itself.

Solis.JPG

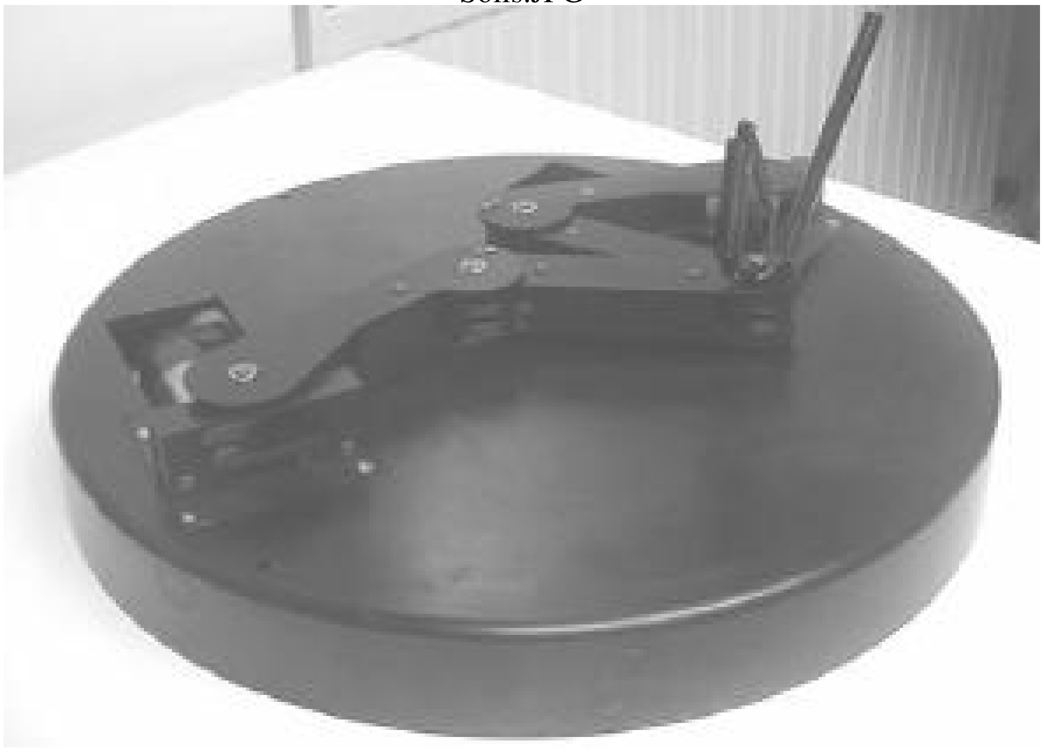


Figure 2.3: Haptic device for writing recognition and correction

2.2.3 Sensorized Gloves

Glove based system represent one of the most important application and research field in gesture recognition[LDD]. They consist in a glove the user can wear equipped with an array of sensors. Some gloves have some kind of force feedback, so they can be considered a kind of haptic device. The first experiments on sensorized gloves started in the '70s. The earliest devices where usually application specific, mounting few hardwired sensors. The main differences between gloves produced nowadays are in the kind of sensors, the way they are connected to the hand (supported by a cloth or directly attached to fingers), the kind of data gathered. Commonly used sensors are piezoresistive, optical fibers (which change the light intensity they carry when bended) and hall effect sensors. The usually retrieved data are hand joint positions, fingertip position or some mix of those.

2.3 Codebook Definition

This section introduces briefly a general concept of codification and analyzes the advantages in using it for this application. Several method described in literature are then presented and compared. The encoding is a sort of feature extraction from raw input data. Codification can be seen as a pre-classification operation (the symbol is the label of a class including the input element) or as quantization (the symbol can be decoded obtaining an element in the domain of input elements). Even if the codebook in this application is used just to encode input data, the concept of quantization will be referred in this chapter when needed to take some conclusion or when, traditionally, the algorithm explained is referred as a quantization algorithm. The codebook can be manually produced on the basis of a deep knowledge of the problem or with adaptive methods. The first method is applicable just when data have a simple and easily understandable structure. Human gesture complexity would require a too large set of rules to be explicitly listed (coherently!) and usually difficult to be defined. Moreover the encoding rules would have to be written again in case of modification/improvement. Hence through the chapter only adaptive methods will be considered.

2.3.1 Precision and Complexity

Using a codebook leads to two simplification of input data:

- Quantization;
- Complexity reduction;

The former consist in representing the input data with a finite set of symbols, the latter in reducing the number of data series taken in account.

Quantization is associated to a loss of precision. Quantization is hence considered a non conservative method. Even if in the application described the codebook is used just to *encode* the data in a useful way, in order to quantify it also a *decoding* stage should be considered, so that the precision achievable encoding and decoding the data is the quantization precision. The measure of precision loss introduced by a quantizer (that represent encoding and decoding together) is the *average distortion*:

$$D = \sum_{k=1}^{L} \int_{m \in P_k} d(m, v_k) f_M(m) dm \tag{2.1}$$

Where L is the number of symbols in the codebook, P_k is the subset of the input data domain M associated with the k^{th} symbol by the quantizer, $d(m, v_k)$ is a measure of distance between an element $m \in M$ and the value v_k used to represent all the values $m \in P_k$. A quantizer is optimum when minimizes D for a given $f_M(m)$. The distance $d(m, v_k)$ could be defined in various way for example the commonly used

$$d(m, v_k) = (m - v_k)^2 \tag{2.2}$$

When using D is referred as *mean-square distortion*. It is anyway to point that representing a set of input vectors with a symbol is not just a simplification, but also a first step of the generalization performed by the classification system.

Reducing the number of data series brings advantages in term of speed having the recognition system less computation to do[1], and of efficiency having less parameters to adapt in the recognition system: the number of parameters affects not only the duration of the adaptation phase, but also the quality of the results. An adaptive classification system with too much parameters can happen to be affected by *overtraining* that is an exceeding adaptation to the examples given, with loss of generalization. A theoretical explanation of this intuitively possible phenomena is given in [Vap99], where the risk of making a classification is shown to grow with the scattering capacity of a classifier. This comes in expense of a certain loss of information. The series of the reduced set should be a certain combination of the original set of values build to maximize the information brought. An indicator of how much information is preserved reducing the complexity could be the percent of the total variance of the original series brought by the ones of the reduced set:

$$V_{\%} = \frac{\sum_{S' \in R} var(S')}{\sum_{S \in O} var(S)} \cdot 100\ \% \tag{2.3}$$

Where R is the reduced set, and O the original set of data series. As example consider the following graphic showing how variance is distributed in the original and in a transformed set. The transformation is performed through PCA, or *Principal component analysis* (see appendix C), notice how a reduced set featuring the first 3 transformed series is capable of holding more than the 90% of the original variance. The data used in the example are sampled by the sensorized glove used in this work (see the chapter 7 for details) while performing movements with the left hand.

2.4 Quantization and Classification

2.4.1 Self Organizing maps

Self organizing maps, or SOMs, are classification networks mapping the the input vectors to an output space with lesser dimensions. SOMs are so called because they are trained without supervised algorithms. A peculiarity of this kind of

[1]Of course data encoding should be simple and fast.

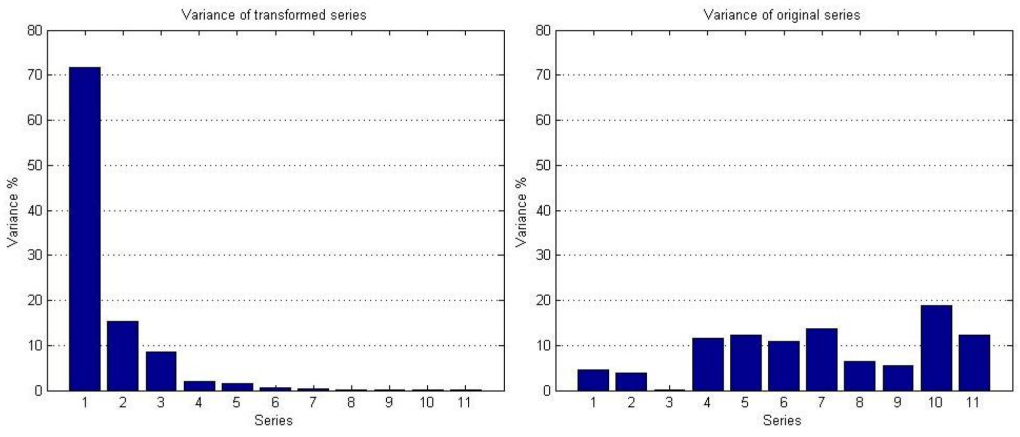


Figure 2.4: Variance of series produced with PCA

network is that it is capable of keeping topological informations about the input vector. This mean that near input vectors are associated to near neurons in the network. Near in this context is meant in the sense of a norm (usually the euclidean norm) for the input neurons, while for the network neurons a predefined topology is given. Every neuron i of a SOM has an associated vector with the same dimension of input vectors, the *weight vector*[2] w_i. When a vector is presented to the network an activation value for every neuron is computed, usually the negative euclidean distance between the input and the weight vector. The neuron with the highest activation value is considered the winner and identifies the input class. Training is performed giving as input the samples of the training set and modifying the weight of the winner vector to be more near to the given sample $p(t)$, where the integer t is the step of the algorithm.
For example consider the Kohonen rule[Koh90]:

$$w_i(t) = w_i(t-1) + \alpha(p(t) - w_i(t-1)) \tag{2.4}$$

Where α is a coefficient called *learning rate*. The same modify, is applied to the neurons near to the winner, according to the chosen topology (the neighborhood). During the training the neighborhood is made progressively tighter, and $alpha$ is decreased in modulus.

[2]The term comes from the traditional feed-forward neural networks where neuron inputs (net inputs or other neurons output) are multiplied with a weight before computing the neuron output value, that is function of them. Commonly the weight are the adaptive component in the net, so in the case of a SOM, the term weight points that the vector is the adapted during the training. But they are not weight in a proper sense, because computing the activation value shall not include a multiplication with them.

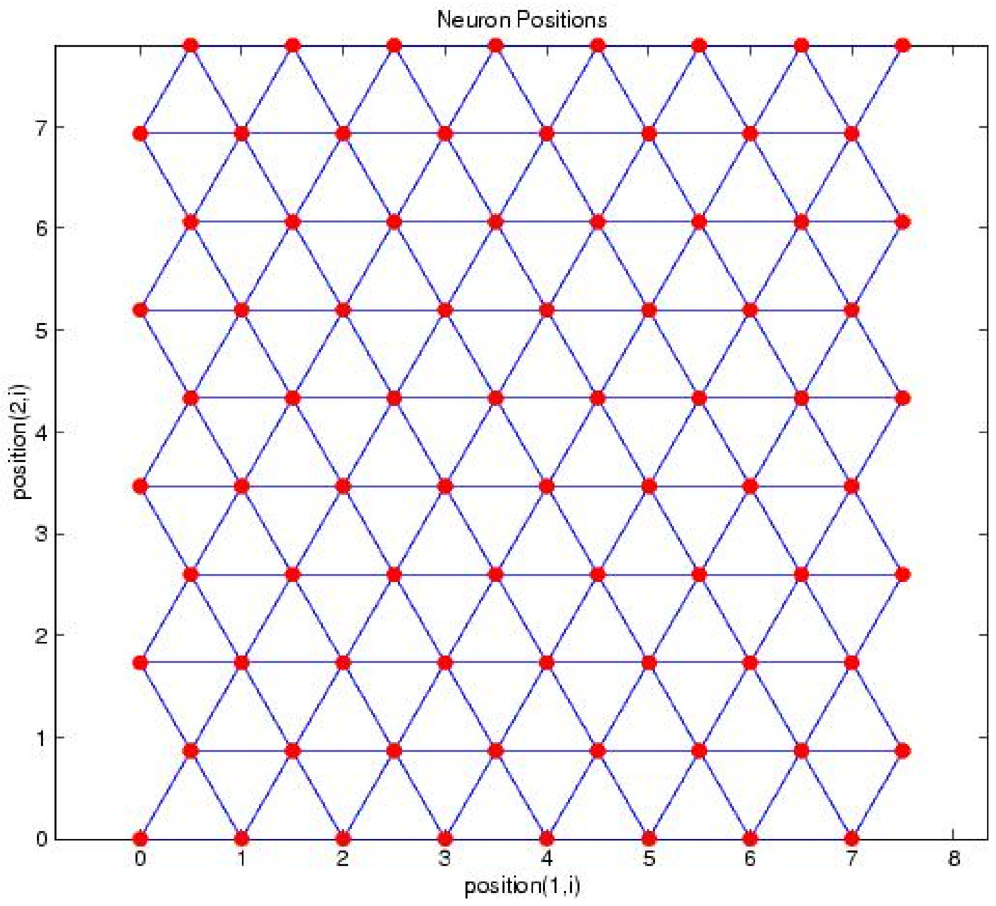


Figure 2.5: A possible topology for a SOM

2.4.2 Bayesian Learning Self Organizing Maps

The model described above does not compute neither exploits any information about probability distributions of the classes of samples given. That can affect negatively the performance in application where input vectors are produced by different probability distributions and the classification success is more important than minimize the quantization error[YA97]. Assuming that the vectors are produced by a mixture of gaussians the parameters θ_i to identify are the a priori probability $P(c_i)$ for the class c_i to have emitted a symbol, the mean vector $\hat{\mathbf{m}}_i$ and the variance matrix $\hat{\Sigma}_i$ for the gaussian representing the i^{th} class. The Bayesian Self organizing Maps implement the identification of such parameters based on the a posteriori probability maximization of the example vector to belong to a class. The winner is the class associated with the distribution more likely to have emitted the input vector. The updating formulas are:

$$\hat{\mathbf{m}}_i(t+1) = \hat{\mathbf{m}}_i(t) + \alpha P(C_i|p(t),\theta_i)(p(t) - \hat{\mathbf{m}}_i(t)) \tag{2.5}$$

$$\hat{\Sigma}_i(t+1) = \hat{\Sigma}_i(t) + \alpha P(C_i|p(t),\theta_i)\left\{[p(t) - \hat{\mathbf{m}}_i(t)][p(t) - \hat{\mathbf{m}}_i(t)]^T - \hat{\Sigma}_i(t)\right\} \tag{2.6}$$

$$P(c_i|t+1) = P(c_i|t) + \alpha[P(c_i|p(t),\theta_i) - P(c_i|t)] \tag{2.7}$$

The posterior probability takes the role of the concept of neighborhood and at every step *all* the neurons are update proportionally with it. BSOM are capable to achieve an optimal accuracy in classifying vectors produced by gaussian distributions.

Another kind of unsupervised networks, exactly the one used in this application, is described in detail in appendix D.3

2.4.3 Learning Vector Quantization

Learning vector quantization (LVQ) is a method to quantize vectorial inputs exploiting a competitive neural network together with a supervised network. Competitive neural networks have several output neurons, just one, considered the winner on a certain basis has a exit value equal to 1, while the other neurons have it null. The index of the winner neuron is a label for the class In the picture 2.6 the LVQ implemented by Matlab is shown as described in the documentation. The architecture of this system is made of two layers: the first is a not supervised competitive network the second is a linear supervised network. The first layer *learns* automatically to cluster input vector: the clustering is based just on the euclidean distance between the input vector and a *weight vector* representing the cluster. The supervised layer is trained later, taking as input the output of the already trained first layer,and the desired label for its cluster (symbol). Notice that the first layer can define more clusters than the desired output labels (both are parameters to set)

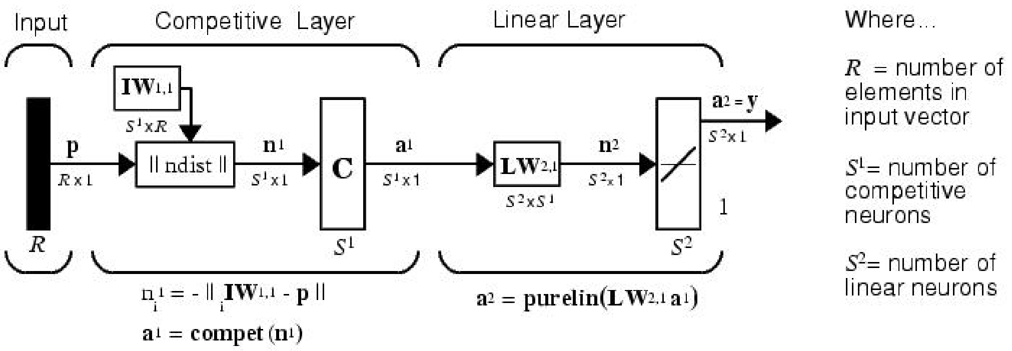


Figure 2.6: LVQ implemented in Matlab

a recognized class included the elements classified in more than a cluster by the first layer. Several training algorithm are available.

2.4.4 Radial Basis Function Networks

A *radial basis function network*, or RBF, is a neural network the neurons of which have an activation function $\phi(x)$dependent only from the distance of the input vector x from a given *center* vector c, so that $\phi(x) = \phi(\|x-c\|)$. The structure of an RBF network as implemented in matlab is shown in figure 2.7, they work similarly to LVQ. The function *purelin* referred in the picture is a linear transformation with trainable parameters. RBF network usually count a bigger number of neurons than other kind of networking performing the same task, but they are quicker to train[Mat04]. Training principles can be both geometric (for example *least square error*) or statistic (for example *maximum likelihood*). Notice that in case $\phi(x)$ is a gaussian function RBF networks become a trainable model of a gaussian mixture. Besides classification they can be used to interpolate functions(also exactly, and computing directly their parameters).

2.4.5 Support Vector Machines

Support vector machines, or *SVM* are classifiers trained in a supervised way, capable of shattering object of the input space into two classes. The main limitations in performing this task with neural networks is that linear neural networks are not capable of separating elements of non linearly-separable classes and multilayer linear networks are complex to train because of the great number of parameters. The SVM are designed to solve these problems. they can be trained with an efficient algorithm and classify elements of non-linearly separable classes. Basically the SVM is a linear classifier that defines the hyperplane separating the two

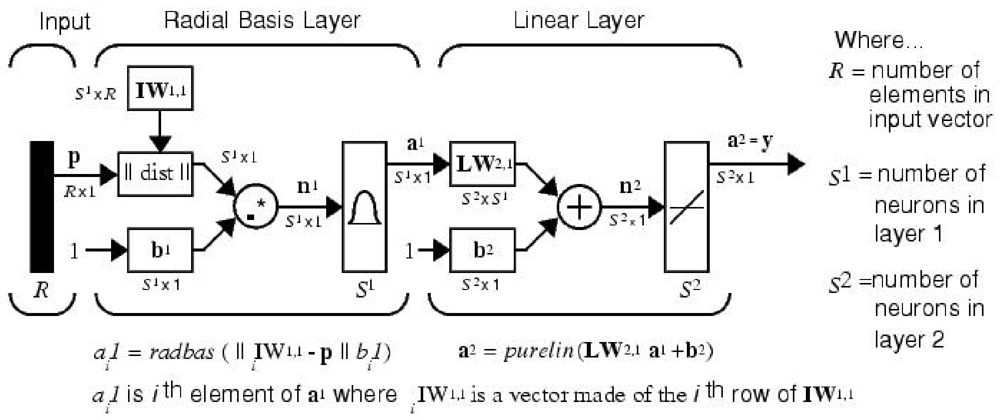

Input
Radial Basis Layer
Linear Layer
Where...
R = number of elements in input vector
S1 = number of neurons in layer 1
S2 = number of neurons in layer 2
IW1,1
S1xR
p
Rx1
|| dist ||
S1x1
n1
S1x1
b1
S1x1
1
R
S1
a1
S1x1
LW2,1
S2xS1
n2
S2x1
b2
S2x1
1
S2
a2 = y
S2x1
a1_i = radbas (|| iIW1,1 - p || b1_i)
a2 = purelin(LW2,1 a1 + b2)
a1_i is i th element of a1 where iIW1,1 is a vector made of the i th row of IW1,1

Figure 2.7: Radial basis function network

classes searching the maximum margin from it and the elements of the training set. Shattering non linearly separable inputs is acheived through the use of kernel functions, as explained in the following paragraph.

Kernel Methods

Non-linearly separable classes of vectors can be transformed into linearly separable classes by an opportune function that maps their elements on an another space euclidean, usually with higer dimension than the original one. Computing the best shattering hyperplane for a SVM involves computing the inner product of the input elements. This can be computed in the space mapped by the function. A kernel function generalize this procedure, being the composition of the mapping from the input space to the higher dimention one and the inner product. Hence it is a function that maps the space of couples of input vectors to reals. Usually it is defined without using an explicit transformation between the two spaces. There are some theorems/rules that defines which fuctions can be cosidered a kernel function. Every algorithm exploiting an inner product can use this method. Besides allowing non-liner shattering with linear methods, the kernel function can also be applied to shatter classes of non vectorial elements, once the kernel function is defined on them.

2.4.6 A comparative experiment

In the article [AC00], three architectures for gesture recognition are compared:

- SOM as codebook, HMM as sequence recognitor;

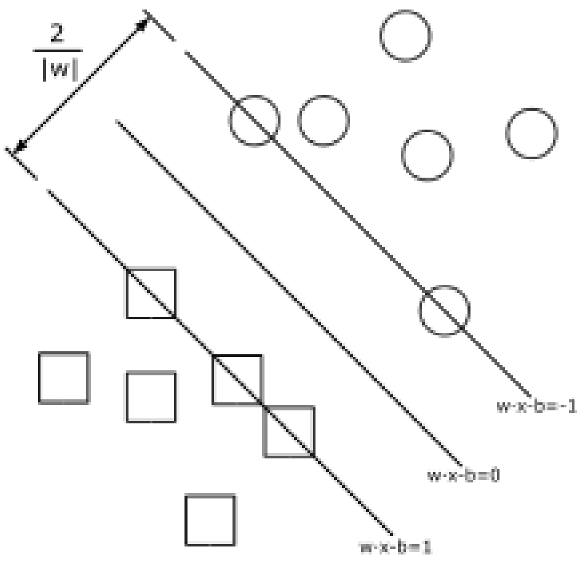


Figure 2.8: Support vector machine optimal hyperplanes and training samples

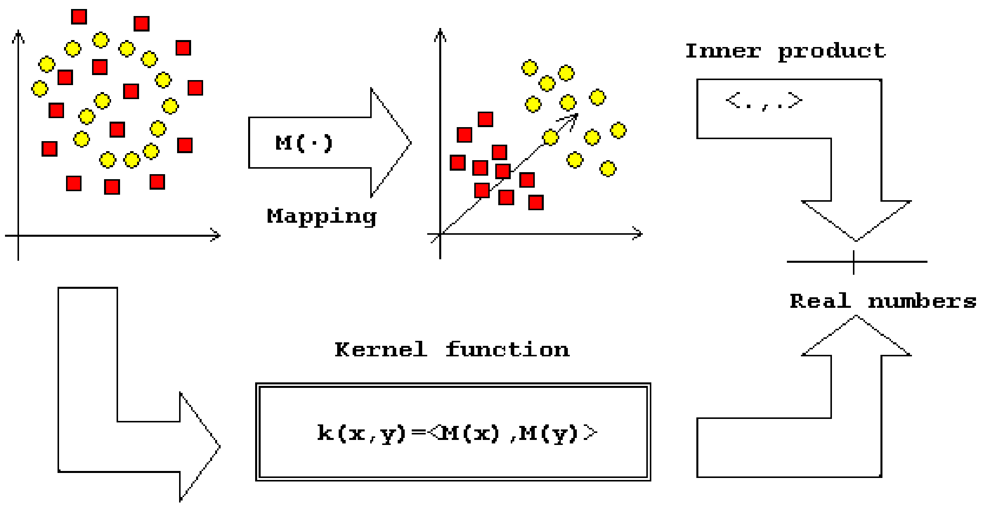


Figure 2.9: The concept of kernel function

- RBF as model for emission probabilities for HMM;
- dynamic time warping;

Dynamic time warping is explained in section 2.5.3.

The results in classifying 5 gestures are displayed in the following tables. The models have been trained with examples build making five people performing 45 gestures.

The RBF outperforms the other methods, anyway due to limited training data[3] is

gesture	SOM	RBF	DTW
stop.	83.6	84.2	76.2
hi r.	85.4	88.9	76.2
hi l.	85.0	88.5	75.0
go r.	85.5	87.2	74.6
go l.	84.2	91.2	77.3

Table 2.1: Recognition Rate in %

gesture	SOM	RBF	DTW
stop.	6.4	5.2	6.6
hi r.	7.1	4.3	7.4
hi l.	7.4	4.7	7.0
go r.	6.3	5.0	6.0
go l.	7.4	4.4	6.1

Table 2.2: % of wrong classifications

not possible to conclude that in general RBF are absolutely best. The HMM have been trained with Baum-Welsch algorithm[4], an improvement in the performance for the hybrid SOM-HMM is expected by the authors, using a principle leading to an higher discrimination power[5].

2.5 Sequences Recognition

2.5.1 Neural Networks

Neural network is a classical approach to pattern recognition. The simplest solution is to feed a classical kind of neural network (SOM,LVQ,feed forward etc.)

[3]It is an article authors admission.

[4]Maximizing likelihood.

[5]That is, in part what is proved in this thesis

with the sequence represented as a vector. This simple solution is limiting: a gesture is characterized by a dynamic and it is not easy to create an opportune vectorial representation of it. There are hence several architectures created to apply the neural networks to not vectorial features [AS] [PF97], in particular for sequences is possible to use a *recurrent neural network* or RNN. In the recurrent neural networks the output is fed back (with one or more delay step) as an input. This makes the network a dynamic system. The architecture in figure very general: computationally equivalent to a *Turing machine*[6]. The way to train a recurrent neural network known in literature are various, as the principles that can lead them are a lot. It is interesting to compare two different choice in training:

- Adapting the network in a classic way considering both the input and the feed-back;
- Unfolding the network through time steps;

In the first case (also implemented in Matlab) the effect of the weight updating on the fed-back values is not considered, this simplification does not create problems in the most cases if a gradient algorithm is applied to train the weight[7]. In the second, more accurate method, a network with several layers representing the same recurrent element at different time steps. The system can be trained with an usual error-backpropagation algorithm, but is to consider that the weight at different time step are the same.

In this application neural network will be used just in the codebook, anyway is important to define their capabilities as comparison term.
There are several desirable features in RNN, that should be guarantee also in the sequence recognition system for the described application:

- good noise rejection;
- high performance in discrimination tasks[Bri90];
- great adaptability to different tasks;
- a wide literature about theory and practice of using RNNs is available;

Besides, there are some limitations that should be overcome in gesture recognition to achieve a good performance:

1. the number of neurons can grow too much to reach a good performance;

[6]It is anyway difficult to know how many neurons/layers a network should include to perform a given task. For example in neural networks are built adding neurons during the training since a target accuracy is achieved.

[7]A second order algorithm can be *too fast* and emphasize the effect of the approximation

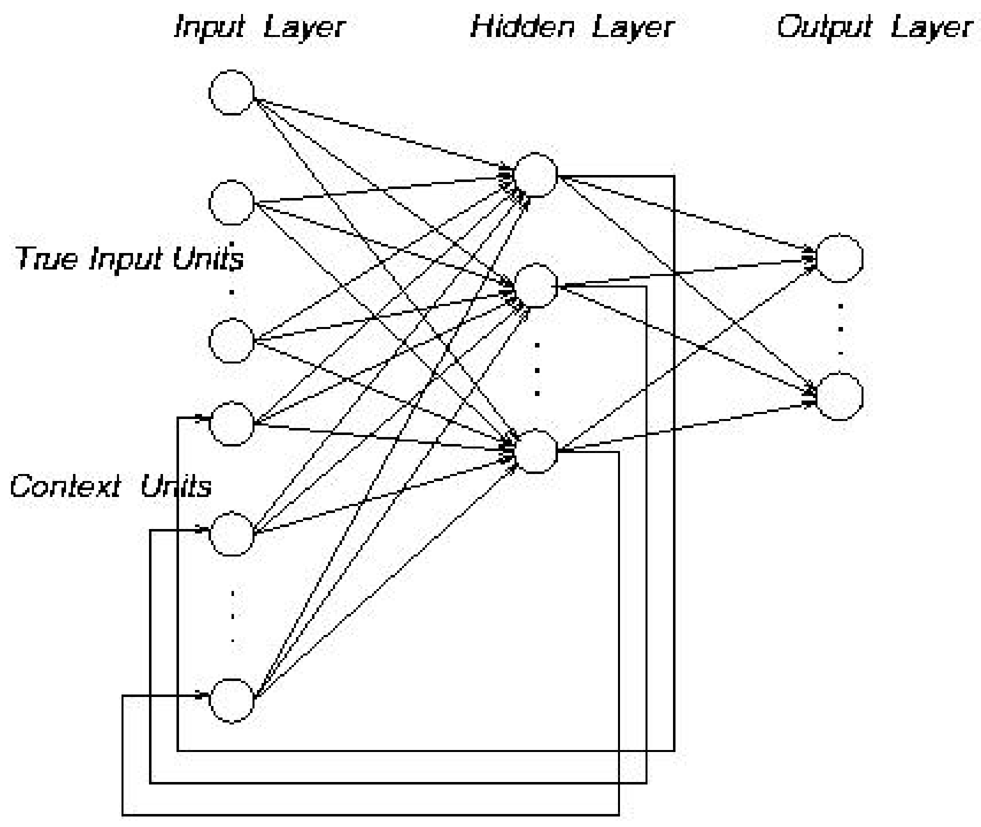


Figure 2.10: A recurrent neural network.

2. results not independent from the speed the gesture is performed with;
3. the generality of the model can make it seem an ”obscure” black-box[8];

In the next section will be discussed how HMMs can avoid this limitations;

2.5.2 Hidden Markov Models and Human Gestures

In this section are discussed the reasons to use HMM in gesture recognition, the current frontier of their application and the problem commonly arising in this context. For an introduction to the model see appendix B.

An HMM is a stochastic dynamic system. In recognition application HMM are adapted through examples to modelize the process to recognize (an HMM for every process). The reason to use HMM are various:

- Defining a model structure, in case it is suitable for the application, even if decreasing the generality can improve the efficiency, avoiding to weight the system with the gathering of the information implicit in the model chosen;
- The number of parameters is relatively small compared to a neural network capable of classifying processes of the same complexity[Mic05];
- HMM parameters have a probabilistic meaning that, integrated with information about the context where they are applied, can be interpreted as objective characteristics of the analyzed process (physics[G.E],events[LR86] etc.);
- HMMs are good models for processes that can happen with different speed, because they can modelize a system that stay in the same state for several steps;
- Although gesture recognition is relatively a new branch of pattern recognition, HMMs have been exploited since the 70’s in application of speech recognition, so a wide literature is available.

The first point is equivalent to say that HMMs are a good approximation of human gestures. There are also more specialized version of HMM that allow state transition just in one direction (the transition matrix is triangular), the *left-right models*[Bis06]. They are employed successfully, besides gesture recognition, written characters recognition[WZ02][JS02] and speech recognition[Cho90].

[8]This psychological argument is controvert (in the author honest opinion, obsolete) but it rises so often that the fear that it could spread over users should be taken in account.

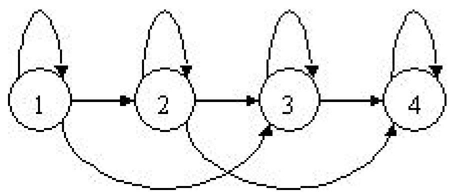


Figure 2.11: Left-Right model

The second point is connected with the first: the less a model is general, less is expected to be the number of parameters. Keeping the model as simple as possible is not just good for training and running speed[9] but also a good precept do decrease the classification error, as proved, with statistical arguments in [Vap99].

The third point is important because meaningful parameters can reveal something about the process under analysis and because sometimes users and developers find unreliable adaptive system that are difficult to understand, as told about neural networks (see 2.5.1)

The forth is a major point of strength of HMMs. Gestures have the evident property of keeping their meaning independently from the speed they are performed with. Anyway the natural capability of HMMs to cope with sequence stretched through time has been found not sufficient in some applications and modified HMMs explicitly taking in account a variable state duration have been proposed[JP], this can also allow to specify a probability density for states time duration that is not necessary the exponential one typical of HMM[10]. The problem is connected with the difficulties in the numeric computation of emission probabilities (formula B.15) for long sequences: Being the emission probability computed as the product of terms smaller than 1 for every sample, it shrinks exponentially to zero with the number of samples. When needed, representing parameters (that are probabilities) with their logarithm can avoid such problems[JS02].

The last point gives a strong theoretical basis and a large set of well known and well tested technical solution to apply to the problems arising in the innovative field of gesture recognition. A lot of problems are shared between speech- and gesture-recognition, hence also the solutions can be shared if opportunely extended.

[9]The classification is anyway usually fast, algorithms are linear in the number of processes considered and the length of the sequence[LR86], the classification itself usually does not happen to be a critical issue in term of speed

[10]The probability for a Markov process to stay present state i at the next state is specified in the model a_{ii}.Hence the probability of keeping the same state for k time steps is a_{ii}^k.

2.5.3 Dynamic Time Warping

Dynamic time warping is an algorithm for measuring similarity between two sequences which may vary in time or speed. In gesture recognition applications it consist in creating a template sequence for every class of gestures, and matching the input sequence with the class associated to the template nearest to it.For example in [MB97] to work with sequences of different length the template is chosen to be as long as the mean of the length of the training set, and it is build as mean of the samples that have the same length of it. Then, when classifying a sequence time alignment through a temporal transform function is performed to make the input sequence to match with the template.

2.6 High Level Description

The human action is usually lead by complex intenctions. A complete analysis of human gesture should hence take in account also an high level description of the task. Historically, an example of high level description is the language grammar exploited in speech recognition. Being gesture recognition exploited in more complex task, more complex descriptions should be exploited. In this section, after a brief introduction on grammar exploited in speech recognition, bayesian programming would be presented as a possible way to modelize the high level structure of the gesture. This will be useful as comparison term for the *ad hoc* solution described in chapter 6.

2.6.1 Grammar and Language Model in Speech Recognition

A grammar is a precise description of a language that is, of a set of strings over some alphabet. In other words, a grammar describes which of the possible sequences of basic items in a language actually constitute valid words or sentences in that language, but it does not describe their semantics. In practice, in a speech recognition application, a grammar could be used to restrict the number of possible recognized words to a smaller set. Usually the set of valid sentences is defined by a set of rules. Languages like XML or semantic webs can be used in practice for this purpose. The theory of grammars itself is beyond the scope of this work. It is interesting to note that the language model can integrate, beside the grammar, also a probabilistic model[Cho90]. For example an HMM can be used to modelize the sentence and retrieve prior probabilities for the low level HMM modellizing the single word.

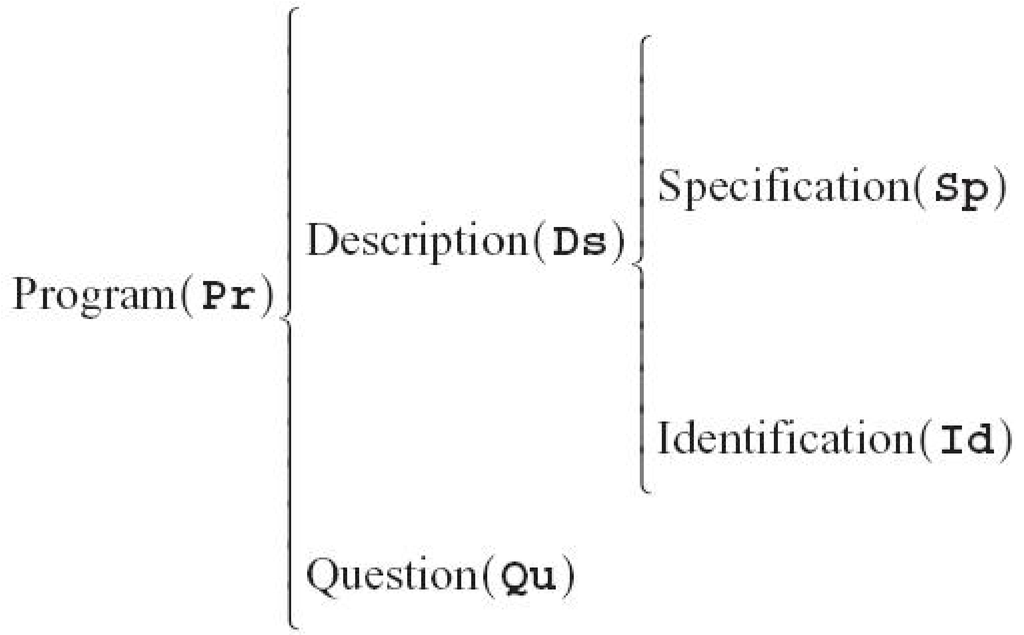


Figure 2.12: Generic structure of a bayesian program

2.6.2 Bayesian Programming

Bayesian programming is a method to create a probabilistic model of a system and to update it according to observed data. A bayesian program includes several fact with associated probability (or probability distributions) and a set of rules to update the probabilities. The bayesian program can answer queries returning the probability of a fact requested by the user. The term bayesian describes an approach that consider probability as the degree of reliability in the possibility of an event, states prior probabilities and adapt them on the basis of observations[11]. Bayesian programming is particularly powerful in applications coping with complex and uncertainly known environments, such as controlling a robot for applications in the real world.

[11]The name comes from the Bayes theorem $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, that in this context is applied to update the probability of the event A once B is observed. Historically speaking, it is not known if Bayes would have used what today we call a ”bayesian approach”

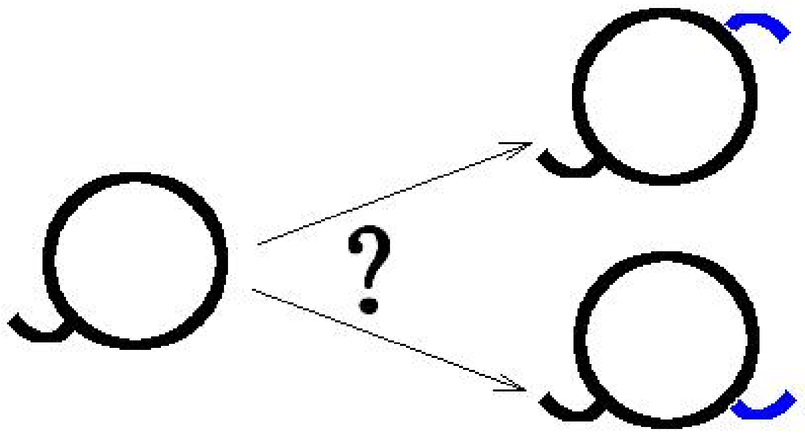


Figure 2.13: A gesture ambiguous until its end

2.7 Problems and Solutions

This section analyzes some problems often arising in gesture recognition and how they have been faced by various author. It will be useful to understand the solutions described in the following chapters.

2.7.1 Segmentation

A gesture has a finite duration that can vary and that is unknown to a recognition system. Segmentation is the division of the input data stream in different segments representing distinct gestures. It is a difficult task, due to the possible variations of human behavior. In some applications [Ros96] HMMs are considered enough powerful to determine the start and the end of a gesture: a gesture can be considered still performed until the probability of the sequence of data registered by the sensors of being produced by it is higher than the probabilities of being produced by other gestures. Is also possible to classify sequences of defined length and consider a gesture to be performed when the partial sequences are classified as part of it. This second solution reduces the speed of the classifier limiting it to give a response only at the end of a sequence[12]. But the real problem with it is that some gestures can be ambiguous until they reveal their nature exhibiting some peculiar aspect. An example could be, if the gesture performed is writing a character, distinguishing a cursive ”o” from a cursive ”a” until the last stroke has been written (figure 2.13).

Segmentation is usually an operation performed at a level higher than the sim-

[12]it can be acceptable if the sequences are enough short

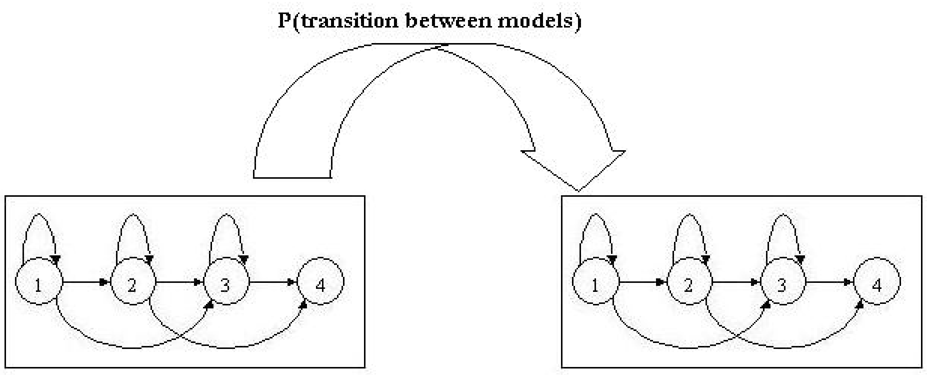


Figure 2.14: CHMM

ple analysis of sequences with, for example, an HMM. This mean that an overstanding model for the whole operation should exist. In application of speech recognition this can be represented by the language grammar[Cho90], in case of gestures in a virtual environment events should work to segment the action.

An interesting approach to the problem is shown in [WZ02] where handwritten English words are recognized through a variant of HMM called *cascade connection HMM* or CCHMM. In this model low level actions (characters or kind of strokes) are represented by left-right HMM. For every model a probability to transit to other models is specified, also more than two levels are possible in this model. It is like having several level of Markov processes, like shown in figure 2.14. Notice that in this case the last element in the left-right HMM has no self recursion being considered the end of the stroke represented. This system is shown to reach 82.34% recogniton rate in when the lowest level has 89.26% accuracy. The main limitation of this approach is that it involves adaptating of a lot of parameters (due to the multiple levels modellized) this mean that a lot of examples are needed (the experiment performed in the cited article needed a training set of 15000 examples).

2.7.2 Two Handed gestures

Using two handed gestures in computer interface is an advanced feature, hence few experimental application have been documented. Anyway two handed gesture recognition seem to be a great resource to produce a natural interaction interface to the computer: some experiments show that two handed interaction is easier to learn, faster and more accourate [AJ05].

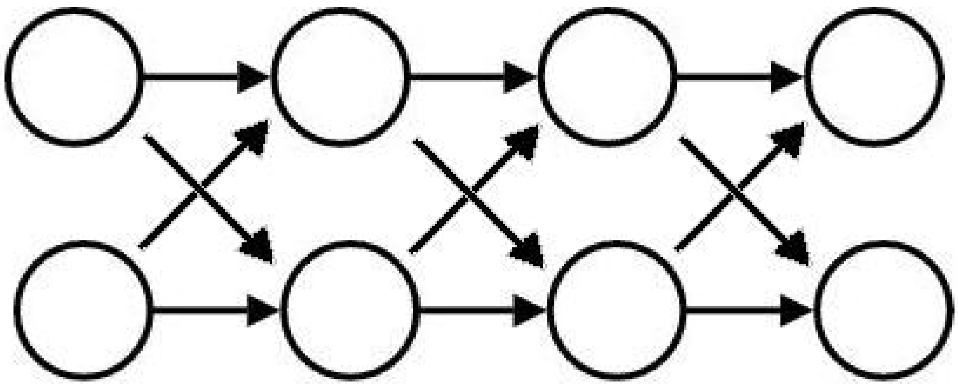


Figure 2.15: Ladder diagram for CHMM

Gestures performed with two hands together introduces the following problems in gesture recognition:

- the hardware should be complicated to gather informations about two hands, without confusing them;
- the combination of two hands dynamics makes the gesture more than twice complicated. This is because all the possible combination of the two movement can come as input. The recognition system can have to incorporate too much internal states to manage this complexity[13];
- the hands could have a different importance in performing a gesture and different degrees of influence one on the other's action. Both these factors can vary between gestures and during the same gesture;

Speaking of experimental applications the first problem is not so important: in the present work the expensive electromagnetic tracking system can mount up to 4 position sensors, so using two hands doesnt increase its cost (neither echonomic or computational), the matter in this case is just to set up two gloves (see chapter 7); in [AJ05] a visual tracking system for two hands exploits coloured gloves, and a visual system without particular markers is presented as one of the future goals in this field. The other points are more complicate problems: the complexity would suggest to create spare models for the two hands, while the mutual dependence requires an unique model for the whole gesture. An interesting solution is proposed in [MB97] where two HMMs representing hands dynamic are coupled by introducing conditional probabilities between their hidden state variables.

[13]Again, gross overfitting would occour.

This model is called *coupled hidden markov model* or CHMM[14], in figure 2.15 is represented the associated ladder diagram (see appendix B) where the two lines of nodes represents the internal states of the two HMM through time, emission is not represented. CHMMs have let no decomposition of the prior probability that might lead to a simple parameter identification procedure. Hence, to train paramenters, an HMM C with states c_{ij} representing a couple of states l_i and r_j of the two original HMMs is produced. At every training step the complex model is adapted and than projected back on the CHMM, this makes the posterior probability increase while the structure of CHMM is respected. The transition probability between two states of C is

$$P(c_{ik}|c_{jl}) = \Psi(P(l_i|l_j), P(r_k|r_l), P(l_i|r_l), P(r_k|l_j)) \tag{2.8}$$

Using a linear Ψ, linear projection will factor the joint HMM into its components, the transition probabilities:

$$P(l_i|l_j) = \sum_l P(r_l) \sum_k P(c_{ik}|c_{jl} P(r_k|r_l) = \sum_j P(l_j) \sum_i P(c_{ik}|c_{jl} \tag{2.9}$$

and the coupling parameters:

$$P(l_i|r_l) = \sum_j P(l_j) \sum_k P(c_{ik}|c_{jl} P(r_k|l_i) = \sum_l P(r_l) \sum_i P(c_{ik}|c_{jl} \tag{2.10}$$

Recognizing 3 gestures, CHMMs in the article acheive an accuracy of 94.2 %. The sample gestures where exercises of *tai chi*, that is a martial art.

2.7.3 Retrieving Quantitative Informations

As told in the introduction the most of gesture recognition applications are limited to symbolic gestures. Hence there is not a wide literature about gathering quantitative parameters from gesture itself. In the figures are shown two sample applications [WB]. The figures 2.16 and 2.17 are examples of recognition of deitic gestures. The user's finger is used as a pointer to choose options and point positions in the space. To work, this application has relatively simple requirements: recognize when the finger is extended (when the hand is closed it can me moved without consequences) and localize the hand. The figure 2.18 shows an application where the shape of an object,a parametric curve[15], is defined with the index fingers and the thumbs. Besides the more complicated recognition task this reveals how the segmentation problem[16] affects the recognition: the shape of the

[14]Note that it can lead to confusion with CCHMM seen in 2.7.1

[15]a Bezier curve.

[16]As seen in section 2.7.1

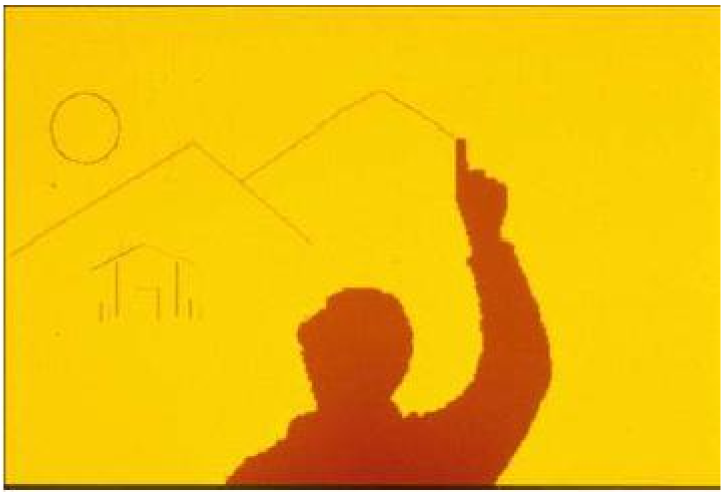


Figure 2.16: Finger painting

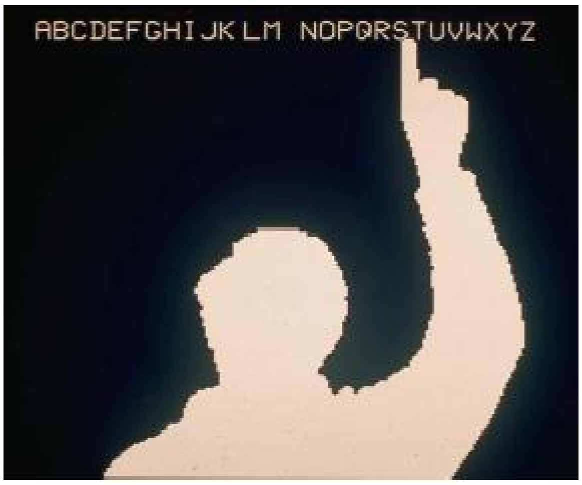
Selecting.JPG 

Figure 2.17: Selecting from a menu

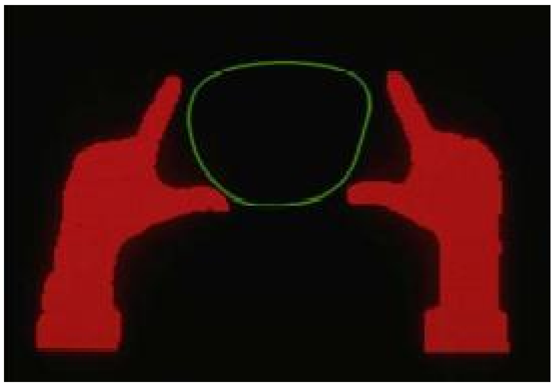


Figure 2.18: Controlling a shape through hand position

figure has to be somehow fixed. Usually in such cases a temporal cue is used, this means that after a certain time with the hands in places the shape is fixed so the user can perform other tasks. In general recognizing a beginning and an end is crucial when the gesture dynamic, and not just the position, brings the quantitative information about the task performed.

Another important thing is the accuracy: the recognition system should be designed to retrieve the geometric information needed with a decent precision. For example, the Glove described in this work (see chapter 7) describes the hand position with 11 degrees of freedom, with non sensors for finger abduction/adduction[17], Hence this movement cannot give a measure. This shows how retrieving quantitative informations is more limited by hardware capabilities than symbolic gesture recognition that, being performed through neural or statistical systems, can work well with approximative or noisy data.

[17]spreading or tighten the hand

Chapter 3

System Overview

This chapter gives a general description of the system architecture designed and employed during the design of the gesture recognition system. The problems handled and the components implemented are mainly the same described in the previous chapter: there will be several references to it to show how classical methods have been integrated in the work and to evaluate the original solutions proposed. Details about important components of this works are given in the chapters from 4 to 7 while remaining details, concerning the implementation are described in appendix A.

3.1 Objectives

The system design is led by the following objectives:

- achieve an high performance with an useful number of gestures;
- take advantage from the environment knowledge;
- Recognizing two handed gestures;
- performing complex operation changing the state of the virtual environment.

3.2 Structure

The system is a mainframe of programs and files, produced with C++, Matlab, and XVR. The functional structure is summarized in picture 3.1, where arrows represent a flow of data and blocks represent elaboration steps. The application works continuously.

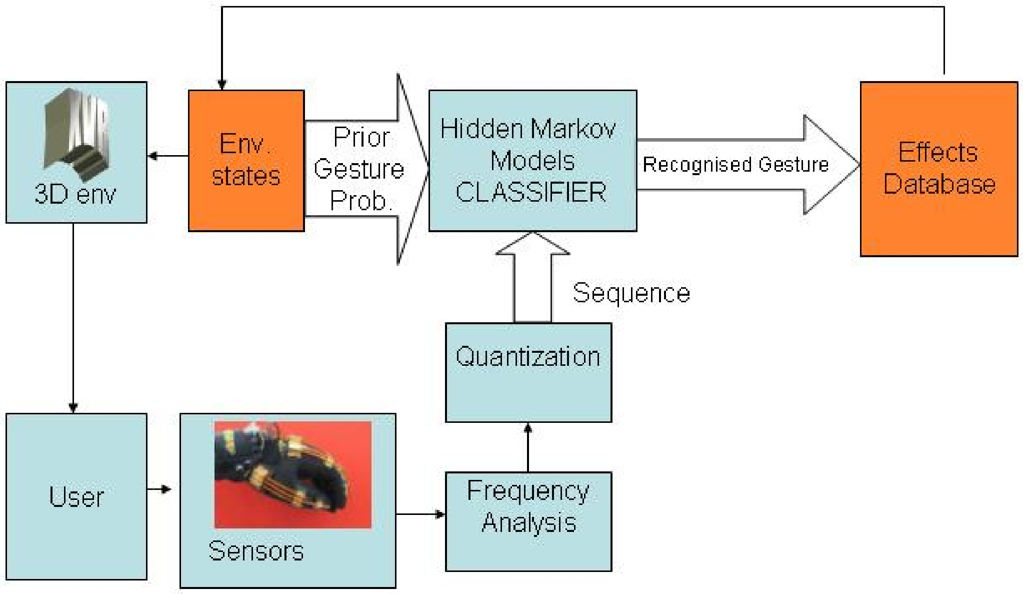


Figure 3.1: System overview

The ”3D env” block represent the application of a 3D motion in virtual reality implemented with XVR. Raw data coming for the glove are analyzed by fourier transform and encoded by quantization through neural networks. Both the operations are performed in Matlab. The classifier runs continuously both producing a classification response and integrating the prior probabilities computed on the basis of the state of virtual environment. Sequence classification is implemented by a library programmed in C++ (see chapter 5), it is compiled as a *dll* and so it can be called by Matlab, the application, or a dedicated program. The ”Effects Database”, representing the effects of the gestures performed on the virtual environment, and ”Env. states”, representing the state of virtual objects are part of the logical description of the environment implemented by a dedicated library (see chapter 6). Producing prior probabilities is the way to aid the recognition system exploiting the environment knowledge. The price to pay is that this knowledge should be explicitly formalized and, at this stage, it should be done besides the geometric model of the 3D virtual environment[1].

This modular structure is designed to be easily adaptable to different applications. The use of Matlab, for signal processing and codebooking is decided with the same aim, anyway, once the application is defined all the components can be compiled and integrated in the same program.

[1]The aim of this work is mainly to test the efficiency of this solution. An application based on it should come with tools to define the virtual environment

3.3 Enhancing Performance

As told the section 3.1 two main objectives of this work is to keep the performance high (in the sense of recognition ratio and, in general, of the accuracy perceived by the users) and to exploit the context informations to do it.

In chapter 2 has been explained how sequence classifiers based on HMM (and in general on generative methods) can exhibit a poor discriminative power compared to methods like recurrent neural networks. Hence a discriminative training principle is applied in adapting classifiers parameters (how this works is explained in chapter 5).

Prior probabilities for gestures are computed on the basis of the situation in which the gestures are performed (see chapter 6). This makes possible to compare, at every recognition performed, just a small subset of plausible gestures, making the system scalable to an higher number of gestures. If an information about prior probabilities in known during the training of the classifier, the algorithm described in chapter 5 can incorporate it: the result is a classifier trained focusing on the differences between gestures more likely to be performed in the same context (and hence confused). Notice that when performing a meaningful task every object has really few ways to be used. For example a screw can only be screwed, unscrewed, put in place to perform one of the two or moved; to unscrew it, it should be screwed (at least partially) an to screw it, it should be put in place. considering this reduces drastically the possibilities.

3.4 Complex Tasks

A logical description for the system is not used just to estimate prior probabilities to enhance performance. It works as a model for high level processes. This mean that the task performed by the user can be more complicated than the simple hand movement: for example to assemble an object in the virtual environment the components positions and interconnection should be tracked during the operation.

Logical variable are useful also to gather quantitative informations during the process. The state of object and of the environment in general can define when an operation starts and when it ends. This mean that the task performed can be measured both in the duration and in the movement. The problem of segmentation is in this way solved bringing it to an higher level: fixed length sequences of samples can be recognized at low level while their effect is regulated by the high level description.

Also two handed gestures are handled at high level. The hands are two elements in the virtual environment and the system can infer conclusions from their state/position. Single handed gestures are recognized at low level, at the same

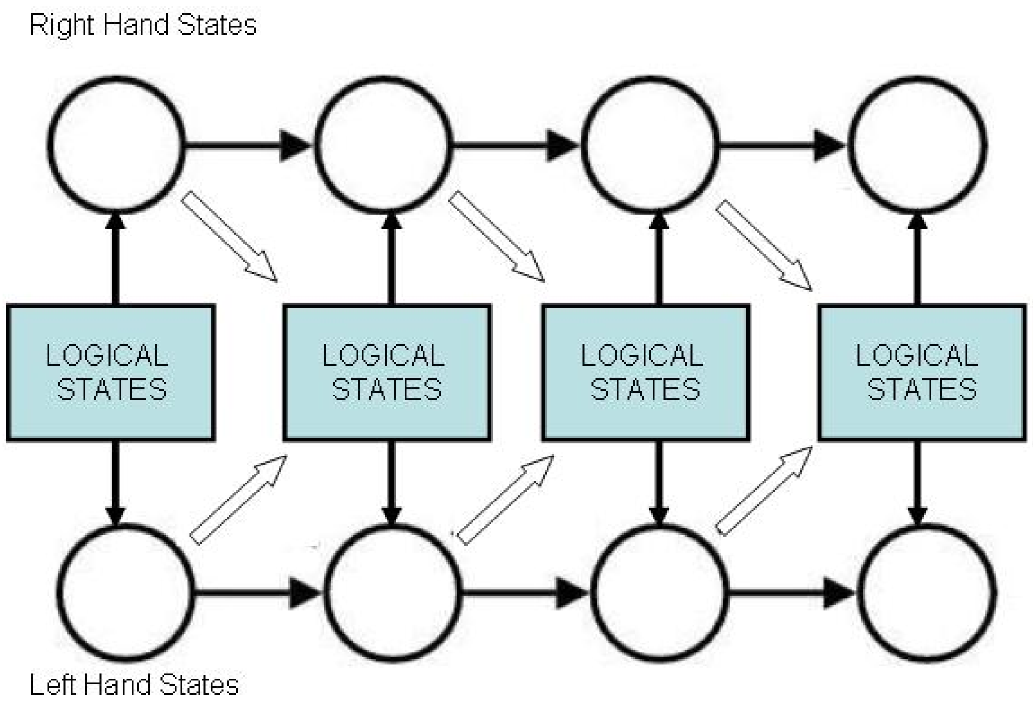


Figure 3.2: Two handed gesture ladder diagram with logical states

time the logical description is updated, this makes the prior probabilities change. In figure 3.4 the model of the two handed gesture is represented graphically in a fashion similar to the one seen in 3.4. Note that in the picture the white arrow does not represent a probability but the effect of the current state on the environment (consequences of the recognized effect). This solution has the advantage that the models are kept simple and so the training of the classifier. The disadvantage is that complex gestures should be described explicitly at high level: this can be acceptable in applications where the objective is the interaction with a virtual environment and there is a logic leading the task;this solution can result tricky if, like in the cited [MB97] (see), the movement itself is important and not the task.

Chapter 4

Codebook Definition and Use

4.1 Components and Parameters

The encoding is performed in three steps:

- complexity reduction by principal component analysis (PCA);
- frequency analysis by fourier transform;
- quantization by neural networks;

The first two steps are linear transformations so the order they are performed is not influent on the result. Making the principal component analysis first simplifies the operation because fourier transform is applied on less series. Neural networks are non linear and they are applied as final step. The encoding process is summarized in figure 4.1, where the a series of 11 signals is reduced to 3 by PCA. Note that once the PCA is applied the other two operations are applied independently on the series. This elaboration requires several parameters to be specified:

- the number of series kept after PCA. This parameter determines the total variance preserved;

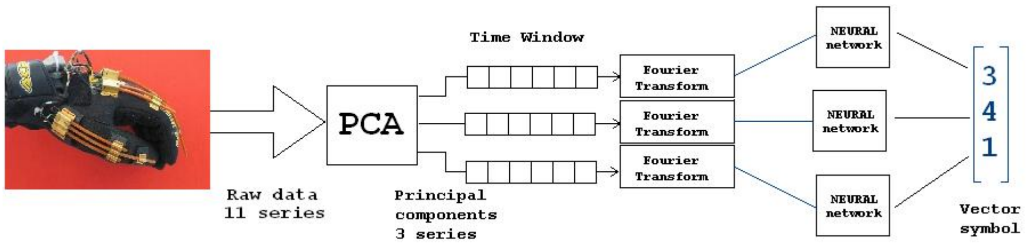


Figure 4.1: Encoding input data

- the time window on which the fourier transform is applied. This is a trade off between the precision and the responsiveness of the system;
- the number of overlapping samples between time sequences;
- the number of classes recognized by the neural network[1]. This parameter define the precision of the neural network and also the number of symbols that the sequence classifier should recognize. If the symbols are too many the parameters of the sequence classifier can grow excessively in number and overfitting to the training set can occur.

This components are described in details in the following sections.

4.2 Principal Component Analysis

Principal component analysis is a procedure to obtain a linear transformation that, applied to data series results in independent series. For procedure details see appendix C. The series produced are ordered on the basis of the variance they hold: using just some of them nearly the whole variance of the system is maintained as shown in the example in section 2.3.1.

Usually PCA is applied to data offline, once a series is fully available. In this work an online encoding is needed. The codebook should be anyway defined in a unique way, so the transformation can not be computed online just on a given time window. The training set is hence used to define the transformation offline and to estimate how many components should be hold to achieve a decent precision. The transformation is than applied online to incoming data. In figure 4.2 is shown the variance of the series obtained with the transformation computed on a training set of five gestures applied on the series of raw input produced by a single gesture.

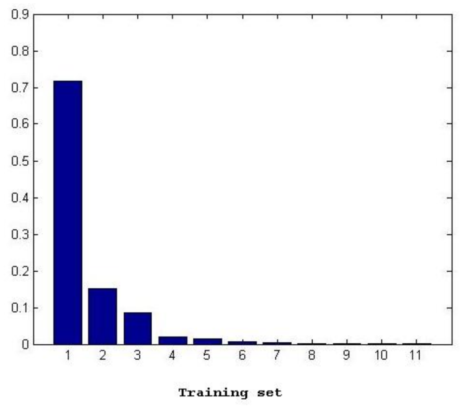
Note that, due to the minor complexity of the single gesture, compared to the training set, less components are needed to describe it.

4.3 Frequency Analysis

A discrete fourier transform is applied on the series before using the neural networks. This let the system focus on the dynamics more that on the position. To make the results independent from the position the first component of the fourier transform, that represent the mean[2] is dropped from the vector obtained. The

[1]The neural network has some other internal settings that will be explained later, this parameter is the more significant at this level.

[2]Or the component at frequency zero



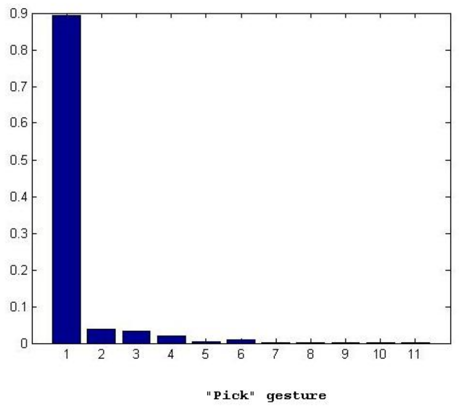


Figure 4.2: PCA for series produced by a gesture

fourier transform results in a vector of complex, and in this form it is used as input for the neural network (In the appendix D.3 will be clear that this introduces no complications).

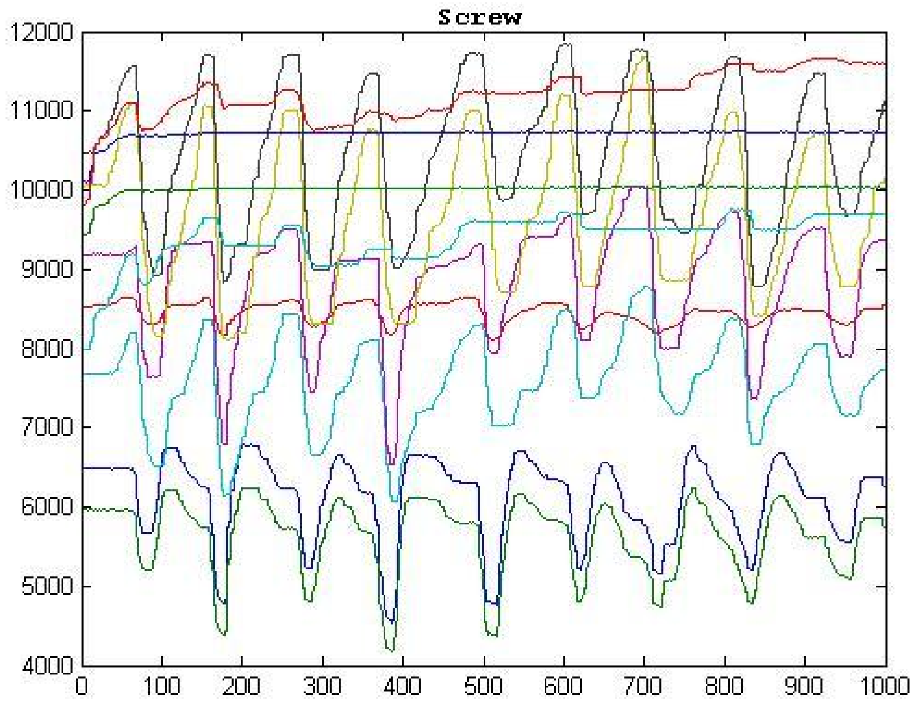
The algorithm applied is the *fast fourier transform* as implemented in Matlab. It has the highest efficiency on vectors of samples with a number of elements that is a power of two, and, in case it is not, a number with small prime factor like 50, the number used in this work.

The fourier transform is conservative, in the sense that all the information is kept (all but the willingly excluded element). This does not mean that the elaboration is not useful: samples of the same gesture can be different represented as sequences of positions and result similar in terms of frequency. The ”weight” of this consideration is not on the neural network, that can be leaner and more efficient.

It is to point tht the vector produced by FFT is complex. The neural network as implemented in matlab can work with complex values as well as real ones. When just the modulus of the FFT is used as input for the neural networks the performance can be strongly degraded. This happen because often the phase is an important element in defining the nature of a gesture. Consider the data stream produced by the ”screw” and the ”unscrew” gestures in figure 4.3. They have almost the same shape but inverted in time; for the property of the fourier transform:

$$F[x(t)] = X(f) \Leftrightarrow F[x(-t)] = -X(-f) \tag{4.1}$$

being the modulus of real signals symetric, the two signals have the same modulus in frequency but inverted phase.



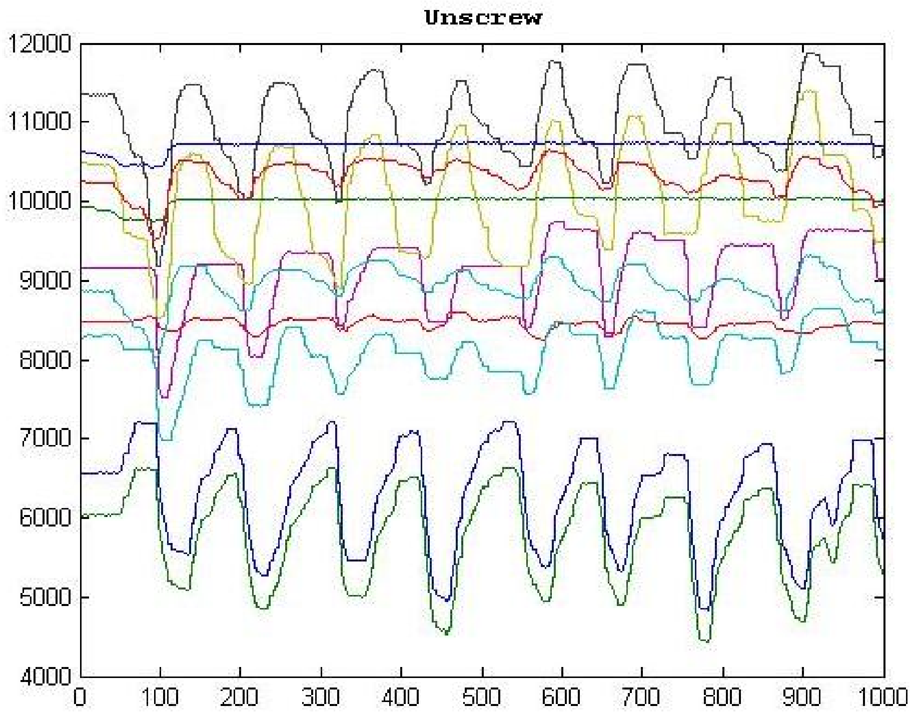


Figure 4.3: Data stream describing screwing and unscrewing

4.4 Neural Quantization

The neural networks are used to finally produce the encoding symbol. The structure and the training algorithm exploited are explained in the appendix D.3. The competitive neural network exploited perform a quantization splitting the input space into several clusters. For every sequence there is a neural network trained separately.

The neural network output is an integer representing the cluster in which the input vector ha been classified. The numbers have no particular meaning, and no relation one with the other.

During the training phase the same training set is used both for the neural networks and the HMMs. The neural networks are of course trained before the HMMs because they are needed to produce the symbols. The classification is performed minimizing the quadratic error on the training set: this is possible arranging the clusters to have the minimum possible error on the most frequent input vectors.

4.5 Information, Precision and performance

As anticipated in the previous chapters, the codebook is used just to translate the input into symbols and never to do the opposite. Anyway it can be interesting to check the precision of the codebook to evaluate its efficiency.

The principal component analysis, discarding components, is a projection of the series on the hyperplane having the first N principal components as generators. These generators are independent, and hence orthogonal. Going back to the first representation by the applying the pseudoinverse of the transformation the quadratic error is the squared modulus of the dropped components multiplied for the coefficient associated to them. The variance of the principal components, having they zero mean, is the squared modulus itself. Hence the precision is directly proportional to the amount of variance kept. Of course it is true for the training set on which the transformation is computed. Experiments shows that once the number of components hold is enough to save the 90% of the training set variance there are no advantage to increase the number of series in term of classification success.

The fourier transform does not introduce any loss of information. It just helps the analysis. Experiments show that the efficiency can decrease from 90% to less than 60% if the mean is not cut off.

The neural networks can be as precise as needed, just increasing the number of neurons. The matter is that HMMs parameters grow linearly with the number of output symbols. In figure 4.4 is shown how this produces overfitting when too

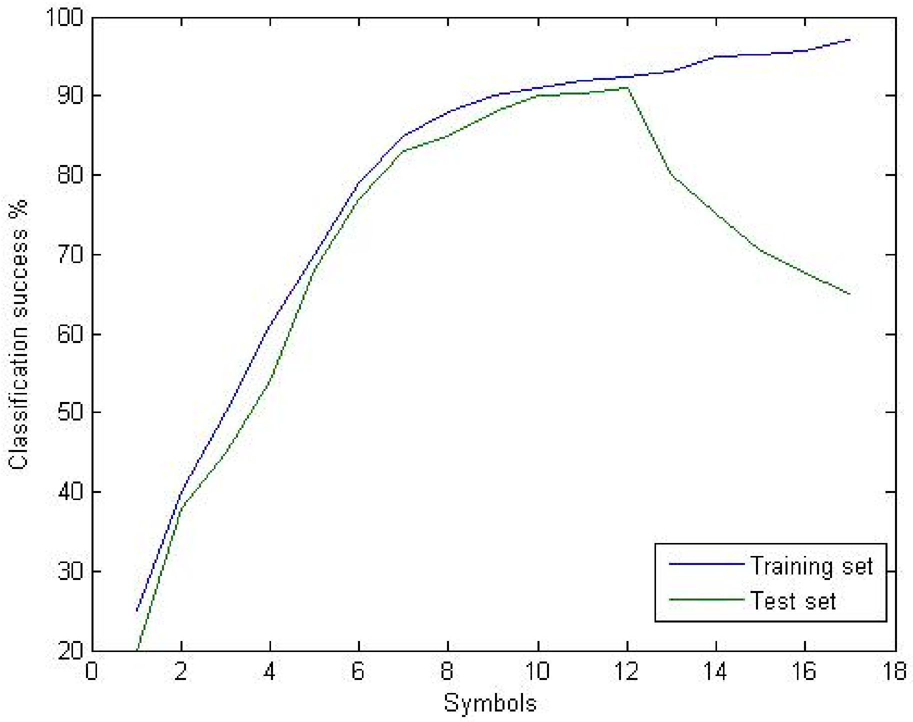


Figure 4.4: Overfitting

many symbols are used.

Chapter 5

Sequences Recognition System

This chapter describes the peculiarities of the implemented classification method, Hidden Markov Models are described in appendix together with the notation exploited.

5.1 Classifier Structure

As exposed in the Overview, the classification of sequences is implemented through a collection of Hidden Markov Models (HMM) with vectorial output. Each model represents a class of inputs. The classification is usually performed by the assignment of a given input sequence to the class associated to the HMM with the highest probability to emit it, as described in the formula:

$$C = maxarg_i \left(P(S|M_i) \right) \tag{5.1}$$

Where C is the class given as output for the input sequence **S**. In this application, for reasons that will be explained in this chapter, the classification is performed by a different formula:

$$C = maxarg_i \left(P(M_i|S) \right) \tag{5.2}$$

By this formalism, that represents the emission probability as a conditioned probability, the sequence is considered as the evidence produced by the gesture performed. Every gesture M_i is associated to its own probability

$$P(M_i) \tag{5.3}$$

That is supposed to be given *a priori* and not to be dependent on parameters[1]. The probability of a sequence S to be observed (namely, produced by the models

[1]This makes sense considering P(M_i) a concept of higher level respect to P($S|M_i$). For example, when recognizing gestures, the probability of a sequence to be observed during a gesture is related to movement coordination (low level), while the probability of a gesture to be performed is connected to the task performed (high level)

featured by the classifier) is

$$P_\theta(S) = \sum_i P_\theta(S|M_i)P(M_i) \tag{5.4}$$

where the subscript θ marks the probabilities dependent on the parameters, represented by the vector θ

5.2 Training Principles

Training[2] the parameters of an HMM to maximize the emission probability of a given sequence is a well known problem for which several algorithms are proposed in literature. Maximization of likelihood is the basis of the most of them. But, while representing a well known and (relatively) easy principle to match the model to the distribution of examples, is not optimal for classification problems in the general case, leading maximum likelihood to an optimal classifier when the model chosen for probability distributions (in this case HMM) has the same structure of the process that actually produces samples to classify, and the performance of a classifier is not affected directly by the accuracy the probability distributions of sequences classes are identified with. The important matter is the relative probability distribution between classes. Hence the principle of Maximum Mutual Information (MMI) is followed[Cho90]. The Mutual Information between a sequence S and the model M_i associated to the class i is:

$$I(M_i;S) = \log \frac{P_\theta(S,M_i)}{P_\theta(S)P(M_i)} = \log P_i(S|M_i) - \log(P(S) \tag{5.5}$$

The aim of the training is to maximize the Mutual Information between the given sequence and the model representing the class which the sequence is supposed to be classified in.

5.2.1 MMI in the context of sequence classification

Before describing the algorithm performing the training, a short explanation of Mutual Information is to be given, to understand how its maximization can aid classification performance.

Mutual Information, in the form defined above, is a measure of the discrepancy between the probability of the coincidence of observing the sequence S and of perform the gesture described by M_i, given their joint distribution versus the

[2]The term, common in the field of neural networks, is used here with the meaning of "model parameters adapting based on examples"

probability of their coincidence given only their individual distributions and assuming them independent. Usually Mutual Information is defined between probability distributions, and the form used here is referred as Pointwise Mutual Information. Notice how the value is equal to 0 in case of independent variables, grows with $P(S|M_i)$ and decreases with P(S). P(S),as defined in 5.4, is the sum of the emission probabilities of S by all the model in the classifier, weighted by the probabilities $P(M_i)$. Exploiting MMI is hence like to train the whole system to make every single HMM *to have an high probability to produce in output training sequences belonging to the class they represent, and low probability to emit training sequences belonging to other classes.*In the trade-off between the two goals the former is achieved also by ML (it is not surprising that the first term in the equation 5.5, $P_\theta(S)P(M_i)$ is the likelihood itself.), while the latter is peculiar of this method. Although creating a model for each class this MMI-trained classifier should be considered a discriminative rather than a generative method.

It could be useful to notice another interpretation of MMI,considering again the definition[Bri90]:

$$I(M_i;S) = \log\frac{P_\theta(S,M_i)}{P_\theta(S)P(M_i)} \tag{5.6}$$

terms could be arranged to put in evidence the term $P(M_i)$, not dependent on θ.

$$\log P_\theta(M_i|S) - \log P(M_i) \tag{5.7}$$

Finding the MMI over θ is equivalent to maximize $P_\theta(M_i|S)$, or adapt parameters to *enhance the probability that,* ***if*** *the sequence S is observed, it has been produced by the process M_i* it is in one sense the opposite of ML approach. Notice again that $P_\theta(M_i|S)$ is a property of the whole classifier rather than the single HMM.

5.3 Training Algorithm

MMI approach could not be implemented by an expectation-maximization algorithm, like Baum-Welsch for ML. A gradient algorithm is therefore exploited. There are two kind of constraints to take in account, which are implicit in the definition of probability itself:

- all the entries of HMM characteristic matrices should be positive and less or equal to 1;
- the sum of the elements on the rows of the transition matrix should be equal to 1.

There is also a practical matter to be considered: MMI is, as told, a principle that applies to the whole classifier, this mean that every HMM in it should be trained on the basis of all training samples considered, this leads to increase the training time quadratically with the number of classes considered[3]. The performance should greatly decrease considering a wide set of classes and sample, as needed in non trivial applications. There are several solutions accounted in literature: for example an hybrid ML/MMI method, or focusing the training on sequences considered likely to be confused by the classifier. In this work the latter method is implemented, as described further.

5.3.1 Gradient Computing

Computing the gradient of MI is the basis of this algorithm. It is computed considering each HMM separately. , with positive examples for the class related to it, and a set of negative examples: sequences, belonging to other classes, but easily confused.It does not contradict the fact that MMI considers all the models together, it will be clear noticing how the gradient computation involves global informations like $P_\theta(S)$, and negative examples. It is possible to do so because the emission probabilities related to the HMMs take part to the formula of MI as terms of a sum (see equations 5.4 and 5.5) so the derivative of MI respect to a parameter of a particular HMM is independent from other parameters. The sets of sequences and models are considered in the training by maximizing the function:

$$F_\theta = \sum_i \sum_k I_\theta(M_i; S_k) \tag{5.8}$$

So the gradient is the sum of the gradients of $I_\theta(M_i; S_k)$ for every HMM M_i and every sequence S_k:

$$\nabla_\theta F_\theta = \sum_i \nabla_\theta \sum_k I_\theta(M_i; S_k) \tag{5.9}$$

The function 5.8 resembles the mutual information defined between two probability distributions of sequences and gestures that caused it

$$\sum_i \sum_k P(M_i, S_k) I_\theta(M_i; S_k) \tag{5.10}$$

but the term $P(M_i, S_k)$ is missing. This complete version of MI is unsuitable for the application, because the probability distribution $P(M, S)$ is not directly available, and computing it from the emission probabilities would lead the gradient to lose the property stated in 5.9.It is also to point that the training set itself works as a representation of sequences distribution, in the sense that a sequence S_k associated

[3]For every class there is a certain number of training samples.So, growing the number of classes, the total number of samples grows roughly linearly with it. As well, the whole training set is presented to every model of the classifier (one for every class)

to an high actual $P(M_i, S_k)$ is represented by a large number of sequences similar to it in the training set, so the the formula 5.8 is an approximation of 5.10. The derivative of 5.8 respect to a parameter θ_i could be written as

$$\frac{\partial I(M;A)}{\partial \theta_i} = \frac{\frac{\partial P_\theta(S|M)}{\partial \theta_i}}{P_\theta(S|M)} - \frac{\sum_{\hat{M}} \frac{\partial P_\theta(S|\hat{M})}{\partial \theta_i} P(\hat{M})}{P_\theta(S)} \tag{5.11}$$

where $\hat{M}$ are the generic models and M is the model related to the class wanted to recognize the sequence S. In order to compute

$$\frac{\partial P_\theta(S|M)}{\partial \theta_i} \tag{5.12}$$

the parameters could be evidenced in $P_\theta(S|M)$, for a and b

$$P_\theta(S|M) = \sum_{t=1}^{T} \sum_i \alpha(t-1) a_{ij} b_j(O_t) \beta(t) \tag{5.13}$$

and for π:

$$P_\theta(S|M) = \sum_i \pi_i a_{ij} b_j(O_0) \beta(0) \tag{5.14}$$

Applying the rule of product to compute the derivative for a:

$$\frac{\partial P_\theta(S|M)}{\partial a_{ij}} = \sum_{t=1}^{T} \sum_i \alpha(t-1) \frac{\partial a_{ij}}{\partial a_{ij}} b_j(O_t) \beta(t) = \sum_{t=1}^{T} \sum_i \alpha(t-1) b_j(O_t) \beta(t) \tag{5.15}$$

to apply it to b considering the case of vectorial symbols emitted

$$\frac{\partial b_i(O_t)}{\partial b_{O_{tj}}} = \frac{\partial \prod_{l=1}^{n} b_{O_{tl}}}{\partial b_{O_{tj}}} = \prod_{l=1, l \neq j}^{n} b_{O_{tj}} \tag{5.16}$$

is obtained.

5.3.2 Constraints Management

In some applications with HMMs the constraints are respected applying a renormalization of matrices row after any adjustment of parameters. It happens for example in [JS02], and implicitly when using Baum-Welsh algorithm.

The renormalization consists in the assignment (for a_{ij} for example)

$$a_{ij} \leftarrow \frac{a_{ij}}{\sum_k a_{ik}} \tag{5.17}$$

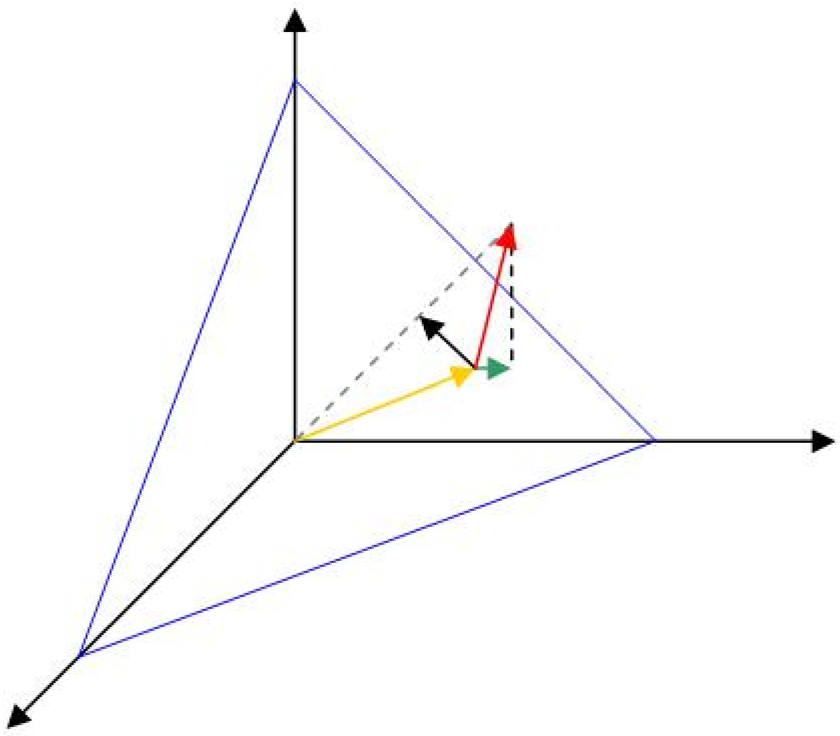


Figure 5.1: The parameter update produced by the normalization and the projection of the gradient

this procedure is unsuitable in the context of a gradient algorithm, as show in the following example. The equation constraints could be expressed as:

$$\begin{bmatrix} 1 & \dots & 1 & 0 & & & \dots & & 0 \\ 0 & \dots & 0 & 1 & \dots & 1 & 0 & \dots & \dots & 0 \\ & & & & & \vdots & & & & \\ & & & & & & & 1 & \dots & 1 \end{bmatrix} \theta = V\theta = \mathbf{1}_n \tag{5.18}$$

they force the parameters to lie on a hyperplan. the normalization 5.17 is equivalent to the substitution

$$\theta \leftarrow \begin{bmatrix} \kappa_1 & \dots & \kappa_1 & 0 & & & \dots & & 0 \\ 0 & \dots & 0 & \kappa_2 & \dots & \kappa_2 & 0 & \dots & \dots & 0 \\ & & & & & \vdots & & & & \\ & & & & & & & \kappa_N & \dots & \kappa_N \end{bmatrix} \theta = K\theta \tag{5.19}$$

where κ_i is the normalization coefficient for any group of parameters belonging to the same distribution, defined so that

$$VK\theta = 1 \tag{5.20}$$

Let$\Delta\theta$ be the vector representing the update performed on parameters at every step

$$\theta \leftarrow \theta + \Delta\theta \tag{5.21}$$

that, for the gradient algorithm is

$$\Delta\theta = \mu \nabla_\theta I(S;M) \tag{5.22}$$

where μ is a scaling factor needed to tune the step length to ensure stability.

Assuming that the μ selected is small enough, the linear approximation of $I(S;M)$ is consistent (this is the basis of gradient descent algorithm) is expected that the operations of parameter updating and parameter renormalization would produce a translation of the parameter vector θ not contrary to the gradient

$$[K(\theta + \Delta\theta) - \theta] \nabla I > 0 \tag{5.23}$$

the update leads instead to a decrease of the objective function.

This is in general not verified, for example when

$$\hat{\theta} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.7 \end{bmatrix}$$

$$\nabla_{\hat{\theta}} I = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \end{bmatrix} \tag{5.24}$$

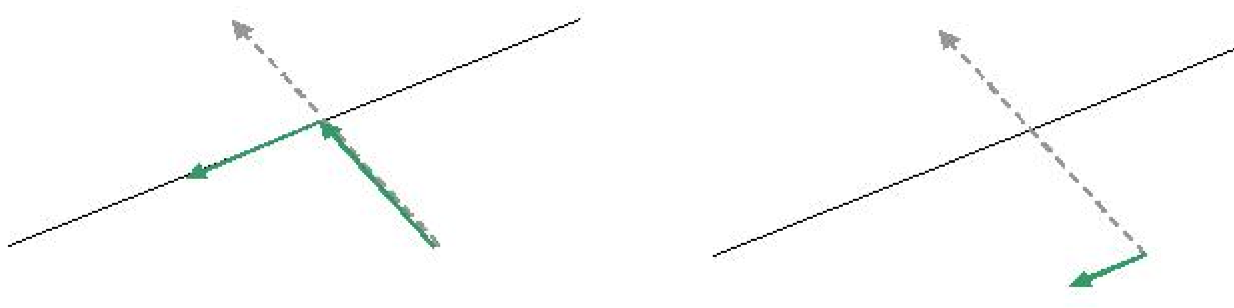


Figure 5.2: Simplified constraints management

the inequality 5.23 is not verified

$$\begin{array}{c}\kappa = 1.5 \\ \mu = 1\end{array} \Rightarrow [K(\theta + \Delta\theta) - \theta]^T \nabla I = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \end{bmatrix} \begin{bmatrix} 0.1 & 0.2 & 0.2 \end{bmatrix} = -0.0033 \tag{5.25}$$

and it is true also for smaller μ.As shown in 5.1 the parameter update produced by normalization (black) and projection (green) of the gradient (red) have two different directions. Their scalar product can be negative.

The more suitable approach is to project the gradient on the hyperplane defined by constraints. The update in this case is

$$\Delta \leftarrow \mu(\nabla I_\theta - V^+ V \nabla I_\theta) \tag{5.26}$$

The constrains expressed as a inequality are managed in the same way, by a projection of $\Delta\theta$. The projection is to be performed only when the constraint is violated. To simplify the implementation, every time the parameters update would produce a constraint violation the $\Delta\theta$ is projected. So θ does not need to lie exactly on the constraints (parameters are not exactly 0 or 1[4]).

5.3.3 Negative Examples

As introduced, not all the samples in the training set are used as negative examples. The sequences considered likely to be confused are chosen as negative example. The confusion margin is quantified by the formula:

$$P(M_k|S_j) > \alpha P(M_i|S_j) \tag{5.27}$$

[4]Actually, parameters are not constrained to be included between 0 and 1 but between 0+ε and 1-ε, where ε is a small value. This is preferred to avoid impossible sequences to exist and make the performance more robust to noise [JS02]

Where S_j is a sample for the class i, k is the class which takes S_j as negative example when the condition is verified and α is a parameter to be specified: for $\alpha = 0$ all the sequences are taken in account, for $\alpha = 1$ only the ones misclassified at a certain step of the training algorithm become negative examples.

As explained in the section 5.2.1 $P(M|S)$ is connected with prior probabilities. This allows to take advantage from high level information in the training process. Batches of samples could be presented with different prior probabilities in order to decide which classes should be affected by negative training.

For example, if a sample S_j for the class i is presented during the training together with a null prior probability $P(M_k)$,S_j will never become a negative example for the class k, being $P(M_k|S_j) = 0$ and so, never greater of $\alpha P(M_i|S_j)$ that is always positive (or at least zero). This is useful because there are some gestures that could occur together in a certain environment and in a certain situation. The performance is so enhanced by a training considering which gestures are likely to be confused, not just for their representation as sequences of symbols, but also their high level relationship with other gestures.

5.3.4 Computational complexity

To compute α and β, that figure in gradient equations, the methods described in **??** is applied[5]. They have quadratic complexity in the number of hidden states. Every algorithm step α and β are computed for every sample (positive or negative), every one of the N models at every sample time. Let T_m be the mean number of symbols in a sample sequences (its duration), S_m the number of sequences (actually not all of them are presented as examples) and n the number of states of the models[6]. The number of operations to be performed are

$$k \cdot N \cdot n^2 \cdot T_m \cdot S_m \tag{5.28}$$

5.4 Mutual Information and Classification

Maximizing the MI is a way to reduce classification errors. Nothwistanding that the two things are not the same: while the whole target function **??**target) is growing, some of its summed terms, related to a sigle example can decrease; it is neither sure that an increment in an term of the target function changes the classification result. It can happen for when the target function could be increased making the terms related to *already well classified* examples at expense of the classification

[5]see appendix B for details

[6]It is possible that different models have different number of hidden states, this is a very particular case and will not be analyzed

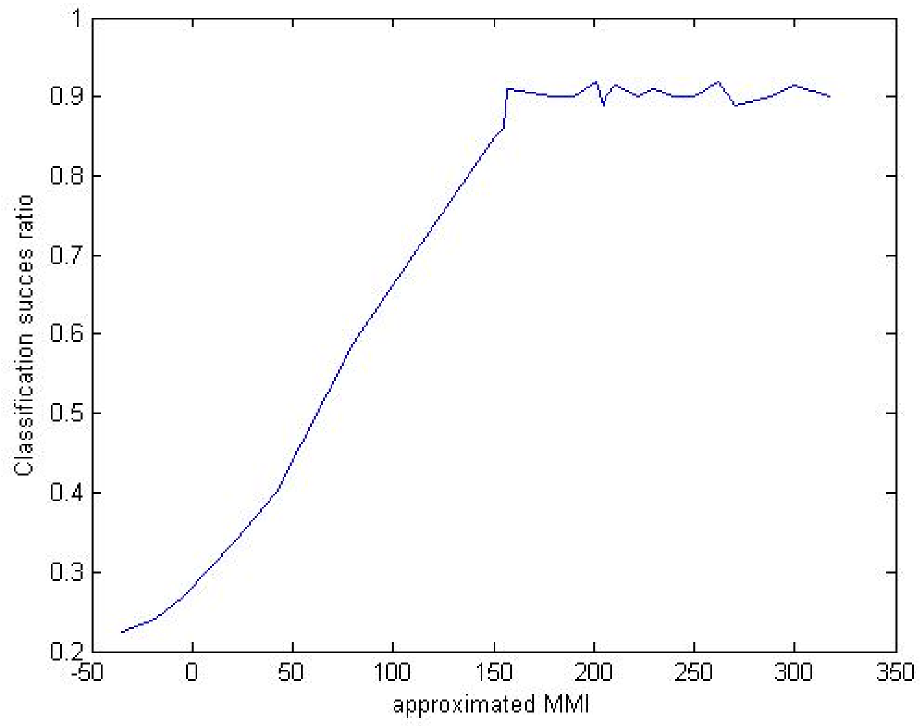


Figure 5.3: Target function and accuracy

of other terms[7]. In figure 5.3 is plotted, as example, the classification accurancy versus the target function. The data are related with a set of sequences obtained encoding hand gestures, considering just finger movements (and not hand orientation).

[7]The rules applied to create the negative training set, exposed in 5.3.3, act also preventing this to degrade too much the performance.

Chapter 6

Logical Environment

The logical description of the environment is implemented by a new inferece algorithm. Routines implementing the inference engine are thought to work in real time and so optimized to retrieve the logical value of a preposition about the environment very quickly. A particular language is used to describe a logical model of the environment: objects, classes of objects and proposition about both objects and classes can be defined in a fashion that represents a simplification of a complete syntax for predicative logic. Fuzzy logical values are exploited. This makes quick and efficient to handle physical concepts like proximity or alignment.

The term *logical description* represent in this context:

- a set of statements with their truth values;
- a set of rules to compute the truth values of statements from other statements truth values;
- a set of rules to compute the probability of a certain gesture to be performed by the user;
- a set of rules to define the consequences of an action;

In this chapter is given a description of the description language syntax, and of the principles used by the inference algorithm. The section 6.4.3 compare the approach described with other ways to implement a logical description of a system, representing a theoretical overview. Further details about the code (C++) are available in A.3.

6.1 Fuzzy Logic

This sections explain quickly how fuzzy operators are defined in this application. Fuzzy logic allows truth values that range between 1 and 0 inclused. This is useful

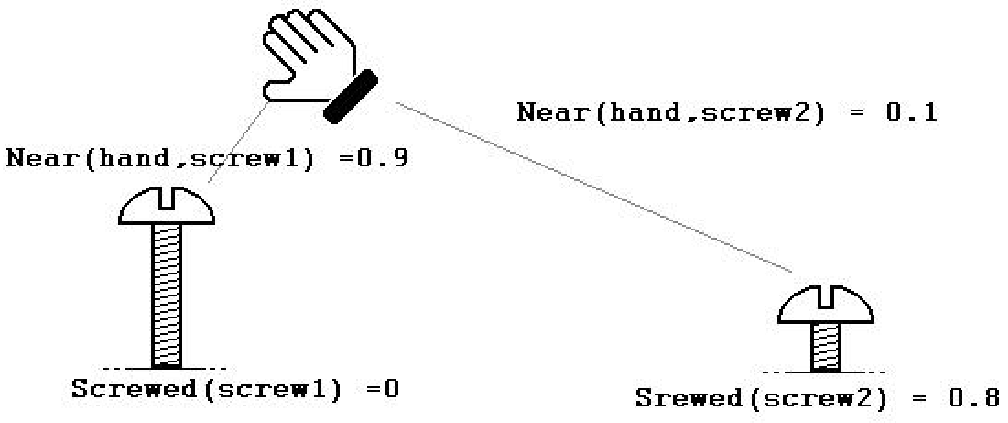


Figure 6.1: A situation in virtual environment and fuzzy values for assumpion on it

to describe concept that are imprecisely defined. Imprecise means that a concept is true with a certain degree, not that it is known with uncertainity, like in the case of probabilistic logic. An example could be the case in which an object is defined ”near” or ”far” from a given point. Even if the object position is exactly known, it could be told that, for example, it is 0.4 near from the reference point. An example model is specified in figure 6.2 where the truth values for ”near” and ”far” are showed as fuction of the distance. Note that these function should be designed as part of the model design (in some application they could be adapted on the basis of examples). Transforming informations in fuzzy values is told *fuzzyfication*.

To work with fuzzy values logic operators should be properly defined. In the described library the following convenction have been employed:

AND operator is defined as the minimum truth value among the operands ones.

OR operator is defined as the maximum truth value among the operands ones. It is not used explicitly in the syntax used to build statements, but it computed in certain cases (see 6.2.5).

Negation operator is defined as the complement to one of the truth value of the negated term.

Implication It is not used in a generic statement but just like a rule to compute truth values from other truth values 6.2.5. This mean that the unknown truth

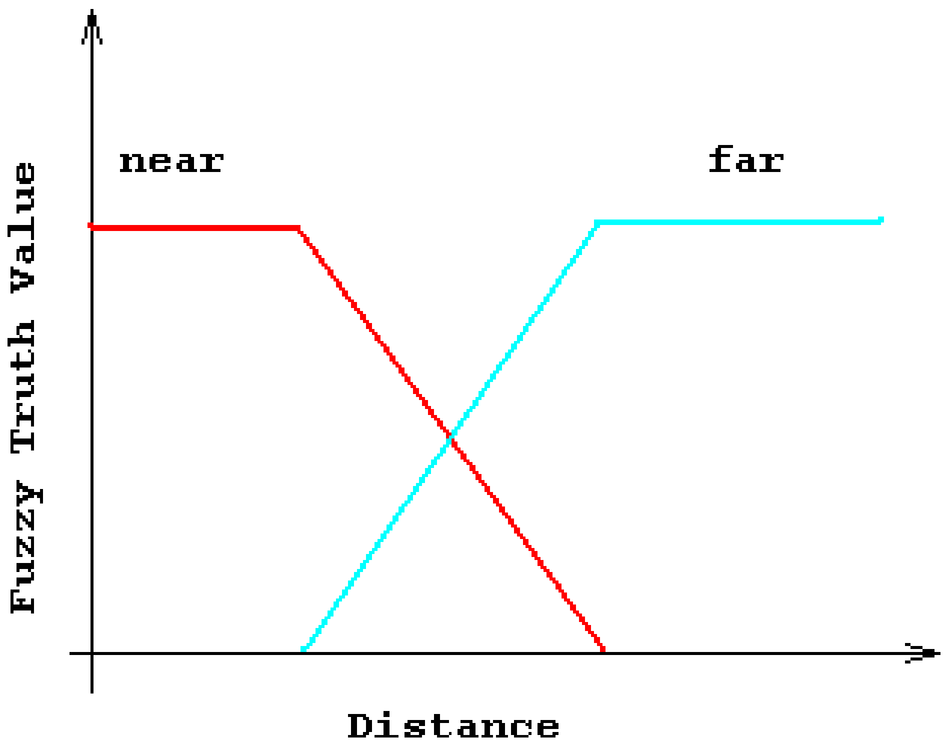


Figure 6.2: Example of imprecise concepts described by fuzzy logic

values are set to maximize the truth value of the implication. Hence, defining the implication as

$$A \Rightarrow B \equiv A \wedge B \vee \neg A \tag{6.1}$$

and considering the definition of AND and OR given it becomes

$$max(min(A,B), 1-A) \tag{6.2}$$

that to compute B given A is maximized:

$$B = argmax_B\,(max(min(A,B), 1-A)) \tag{6.3}$$

Note that, altough associating values between 0 and 1 to statements, fuzzy logic is something different from the probabilities used, for example, in bayesian programming (see 2.6.2). Fuzzy logic does not give a measure of expectation of something to happen or to be true when not yet known, it describes a logical state that lies between two statements. It is deterministic. For example consider the following statement: ”the user hand is near to the screw”. Giving a fuzzy truth value to it would mean to define, having some fuzzification rule, a state of the hand. If the value is 1 or close to it the hand is near the screw, if the value is 0 or close to it, the hand is considered far from the screw. The uncertainity expressed is just ontologic, in other word part of what is described. The fact that what is expressed is known is considered certain.

6.2 Elements and syntax

The logical description of the environment is a collection of statements about the objects represented in the simulation and some rules, expressed in a ”cause+effect” form[1] that are used to build a chain of inferences when computing the logical value not (*or not yet*)included in the description. The elements of the description are *classes of objects*,*functions*, *rules* and *fact*, all of them declared in a special text file or at runtime through a specific function.

6.2.1 Objects Declaration

A class of objects is declared with the command `object` followed by the name of the class. An object is declared with its class name followed by its own name, for example the following code declares the class `screw` and instances two objects `screw1` and `screw2` .

[1]it will be clear soon why the rules are not implications neither clauses in a general sense

```
object screw
screw screw1
screw screw2
```

The class could be referred in statements in order to refer one or all of the object belonging to it,the class itself is not an object referred by sentences, hence it is not term of relations with other objects or have his own attributes. This actually mean that statements like ”all the screws are lubricated” can be part of the environment description, a sentence like ”the screws are four” instead cannot be part of the description because ”lubricated” is an attribute applied to every single ”screw”, ”four”, instead, could only apply to a set of screws and no one of them is obviously ”four”. Notice that only instanced objects are considered, any sentence about ”all the screws” or ”any screw” is implicitly equivalent respectively to a sentence about ”all instanced screws” or ”any screw among instanced ones”. An object can belong to one and only one class.

6.2.2 Functions

Functions associate a fuzzy logical value to an object or a set of objects: a function with its arguments is the simplest statement that could be written.
two examples of statements referring the objects `screws` instanced in the previous example 6.2.1:

```
lubricated(screw1)
near(screw1,screw2)
```

The logical description can include a value of truth for these statements or it can be retrieved using inference rules (how this work will be described in 6.2.5 together with the rules). A function is declared by the keyword `function` followed by the name of the function and its arity, for example: `function lubricated(1)`
`function near(2)`

The arity is the number of arguments, and should be at least one. The given examples, using meaningful words, can lead to confusing interpretations. for example

```
near(screw2,screw1)
near(screw1,screw2)
```

are two different statements, and should have different values! In order to make these statements equivalent, rules should be used to let the application infer one from the other.

Function can also take a class name as argument, this is used to refer to a generic object of that class. for example
`lubricated(screw)`

has the highest truth value among the values of sentences in the form
`lubricated(S)`

where `S` is an instanced object of class `screw` . This is equivalent to an OR, according to the convention explained in **??**. There is also a special argument for the function that is `ANY` that refers to any instanced object.

6.2.3 Statements

A statement is built upon the concept of function. As told a function with its arguments is the simplest statement. A statement is built as a product (AND) of functions with arguments. Arguments could be negated. The logical operator OR is not represented explicitly. an sample statement could be:
`lubricated(screw1) AND lubricated(screw2)`
In case a generic object is referred (through the name of its class) a number could be specified. This allows to refer the same generic object as argument of two functions. For example consider the following set of objects and functions.

```
object pizza
object oven
pizza pizza1
pizza pizza2
oven oven1
function well_done(1)
function inside(2)
```

The two statements below have two different meanings (in the sense that they may not have the same truth value)

```
well_done(pizza) AND inside(pizza,oven)
well_done(pizza#1) AND inside(pizza#1,oven)
```

The first means that a pizza is well done and a pizza, not necessary the same is in the oven, the second means that a pizza is well done and in the oven[2]. Every generic object can have a number, even `ANY` . the negation is expressed with the symbol ”~” put before the element like in:

[2]this is an understandable interpretation, anyway the model is just a set of symbols (the function with arguments) that have some truth values, nothing to do with real concepts like ”pizza” and ”oven”.

```
~well_done(pizza#1) AND inside(pizza#1,oven)
```

The negation applies only to the next element. The operator OR is not allowed to appear explicitly in a statement.

6.2.4 Facts

A fact is an elementary statement (without AND operators) explicitly put in the description, it is declared in the description file with the keyword `fact` , followed by the statement and its truth value (if missing it is considered 1). The facts should refer to instanced objects (no generic object are allowed). The fact could be negated with "~", it just make the list to assign the complementary truth value to that statement.
examples of possible declared facts are:

```
fact well_done(pizza1) 0.5
fact fixed_on(screw1,bolt1) 1.0
fact screwed_on(screw1,bolt1) 0.3
```

6.2.5 Rules

Rules define relationships between facts. They are expressed in a intuitive way in the form of a cause-effect couple. The cause is a generic statement, the effect is an elementary statement (no AND operators allowed). Rules are declared by the keyword `rule` like in the example:

```
rule inside(pizza#1,oven#2) AND turned_on(oven#2) =>
well_done(pizza#1)
```

In this example a "`turned_on` function is supposed to exist. The statement on the right side of "`=>` " (that represent an arrow), can be inferred from statement on the left[3]. Notice that the number "1" is the same for the object `pizza` in the cause and in the effect, this mean that the cause should be applied to the objects of the class `pizza` for which the statement `inside(pizza#1,oven#2) AND turned_on(oven#2)` is true. If more than one rule have the same effect the truth values of all the causes are computed and than the greatest is given as truth value for the effect. This operation is equivalent to apply the operator OR to the causes (see **??**). This make the rules behave like an implication: the effect can be true even if the cause is false if it is triggered from another cause.

[3]It could be different if a value is given through a "fact" or previously computed and kept in memory. It will be explained in detail together with the algorithm exploited to infer truth values from the stored facts 6.4.

A complication arises considering that the effect could be a negated statement. This mean that the effect is inferred with the negated cause truth value. The same statement could appear, as effect, negated and not in two different rules, for example

```
rule well_done(pizza1) => well_done(pizza2)
rule ~well_done(pizza3) => ~well_done(pizza2)
```

can model a context where the condition of a pizza is retrieved from the condition of other two ”pizza” (it could be because the order they entered in the oven is fixed). Notice that the two rules could give two different values for `well_done(pizza2)` . In case of two non negated statements as effect the greatest value was given, but it does not make sense here: if `~well_done(pizza3)` has a truth value greater than the one of `well_done(pizza1)` the negative of the former would be smaller than the one of the latter, this will make the ”weaker” cause be more important than the other.Hence the negation is applied later, after choosing the value of the stronger cause. Traditionally a fuzzy version of implication should be an extension of the classical one, in the sense that is supposed to have the same truth values when applied to crisp values[4]. The difference between a rule as here defined and an implication consist in the fact that the rule explicitly tells how to compute the truth value of a statement (effect) given the value of another statement (cause). An implication is instead a statement itself and can be used to compute the truth values of statements in the cause. Consider the sample implication (crisp version)

$$A \wedge B \Rightarrow C \tag{6.4}$$

If C is false can be deduced that $A \wedge B$ is false, and so, if A is true, B is false, and vice-versa. instead considering the similar rule

```
A(x) AND B(x) => C(x)
```

where `x` is an instanced object, no conclusions about `A(x)` and `B(x)` is taken by the inference engine. Rules are a simplification of the concept of implication, and are used because they bring two advantages:

- they are very quick to analyze at runtime because of their lean structure;
- they are a direct description of how the inference algorithm would compute the truth values, making intuitive to write the system description[5];

If a proper implication is needed in the system description it could be declared in the descriptor file like a rule, adding the symbol ”$” at the end of the line, for

[4]it is not always true in any application, for example the implication proposed by **??** does not satisfy this requirement.

[5]logical demonstrators often are based on a knowledge representation designed to optimize inferences, like **??**

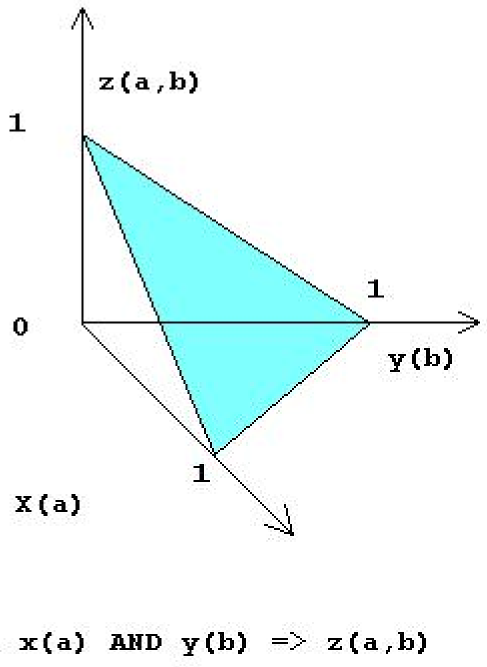


Figure 6.3: A rule: the conseguence has a truth value that is the minimum of the ones of its causes.

example

```
A(x) AND B(x) => C(x) $
```

is equivalent to the implication 6.4 In other words a rule can be considered an "IF…THEN" statement, common in high level programming languages. Notice that in this context the rule is applied only when the truth value of the consequence is asked to the application.

6.2.6 Actions

Actions represent the effects, on the system logical states of gestures performed by the user. These effects are represented with the same syntax of rules. An action is declared with the keyword `action`like in the following example:

```
action screwing:  near(hand,screw #1) AND plugged_on(screw
#1,hole #2) => screwed_on(screw #1, hole #1)
```

Notice the ":" that introduce the effects.

6.2.7 Prior probabilities

A vector of prior probabilities for the several gestures is specified for several logical condition. The logical condition is a statement. Prior probabilities vectors are declared with the keyword `priors` followed by the condition on the same line, then a list of gestures with their probability follows, for example:

```
priors near(hand,screw #1) AND set_on(screw #1,hole)
screwing 0.25
picking 0.25
setting_on 0.5
```

Gestures not listed are considered to have null probability in that particular vector.

When prior probabilities are required by the application a weighted sum of all the prior probability vectors is given. The truth value of the condition is used as weight for the related vector.

6.3 Interface

Once the logical description is loaded from the file, interface functions are used by the application to retrieve logical values and to modify them. In this section the possible operation will be described, focusing on their meaning more than on their implementation. All the function presented are methods of a C++ class that represent the descriptor. More details about the library functions and the structure are available in A.3.

6.3.1 Retrieving truth values

The truth value of a statement can be retrieved by the functions `evaluate` and `revaluate` . Here are the C++ declarations:

```
double evaluate(string statement);
double revaluate(string statement);
```

The argument is a string of characters[6] where the statement is stored with the same syntax used in the file; the returned value is the fuzzy truth value associated with it. The difference between `evaluate` and `revaluate` is that the first just gives the value of the statement in any case, the second computes the

[6]the type `string` , representing an array of ”char”, is not displayed with bold characters in the declarations, because it is not a basic C++ data type. It is introduced by the standard library ”string.h”

value of the statement as if it was not part of the description (if it is not actually present it is the same) and than memorizes it in the description.

6.3.2 Notifying a Fact

A fact can be added at runtime to the description with the function `notify` .
```
void notify(string statement,bool provisional);
```

The fact is specified through the string `statement` . The value `provisional` specifies that the fact is deleted when the method `flushprovisional` is called. Like when declaring a fact in the description file, the truth value should be specified next to the statement.If the fact is already stored in the description its truth value is replaced by the one specified.

6.3.3 Retrieving Prior Probabilities

To get the prior probabilities the function `GetPriors` is used.
```
vector<double> GetPriors();
```

The function takes no arguments because prior probabilities are computed on the basis of the current logical state, like explained in 6.2.7. The data type `vector<double>` is implemented by the C++ standard libraries and is an array of double

6.3.4 Performing Actions

Performing actions is implemented by the function `act` . `void act(string action);`

Only the name that identifies the action is required, the action consequences truth values are computed on the basis of the current logical states. If the consequence of an action is a statement already part of the description the computed truth value replaces the previous one.

6.4 The Inference Engine

The *inference engine* is the set of functions that computes truth values of arbitrary statements from the logical description. In this section the algorithms exploited are described, implementation details are in A.3

6.4.1 Research in the Space of Statements

When the truth value of a statement is required, the following action are performed by the engine:

1. The statement is searched in the description, if it is present its truth value is returned and the research is ended;

2. A rule having the state as *effect* is searched. If found the procedure is repeated for all the terms in the *cause*. The value of truth is than computed and returned.

3. If a value is not found in previous steps, 0 is returned.

Notice that the second step could lead to an infinite looping. consider as example the following description:

```
object weathers weathers weather function cold(1)
function rainy(1)
function cloudy(1)

fact cold(weather) 1

rule rainy(weather) => cloudy(weather)
rule cloudy(weather) AND cold(weather) => rainy(weather)
```

In case that the truth value of `rainy(weather)` is asked, the second rule is used (because the statement is not present in the description) to compute it. The term `cold(weather)` is available in the description so is taken directly. The to evaluate the value of the term `cloudy(weather)` the engine uses the first rule, but this rule needs the value of `rainy(weather)` , so the second rule is used again! An *anticycle rule* is used to avoid the engine to loop forever (and so stop working). A rule can not be used in a node of the demonstration tree that has a parent represented by an application of it[7]. A demonstration tree is the non looping graph that represent the steps performed, nodes represent the application of rules or (the terminal ones) a fact.

If several rules with the same *effect* are available the one with the *cause* with greatest truth value is considered (see 6.2.5). If generic object are referred, using the class name or the keyword `ANY` the engine creates all the particular rules (only referring to instances objects) considering the statement whose truth value is asked. For example,given the description

[7]A rule referring to general objects can be used in the same branch if applied to different instanced objects

```
function happy(1)
function friend_of(2)
object people
people Thomas
people Joseph
people Henry

rule happy(people#1) AND friend_of(people#2,people#1) =>
happy(people#2)
```

a request for the truth value of `happy(Henry)` , will make the system create the following rules

```
rule happy(Thomas) AND friend_of(Henry,Thomas) => happy(Henry)
rule happy(Joseph) AND friend_of(Henry,Joseph) => happy(Henry)
rule happy(Henry) AND friend_of(Henry,Henry) => happy(Henry)
```

6.4.2 Representing implications

An implication is different from a *rule* because it gives the possibility compute, not just the effect from the cause. Implications are implemented creating several rules to describe the possible deductions. For example the declaration:

```
rule happy(Thomas) AND friend_of(Henry,Thomas) => happy(Henry) $
```

Is represented in the description by a set of rules

```
rule happy(Thomas) AND friend_of(Henry,Thomas) => happy(Henry)
rule ~happy(Henry) happy(Thomas) AND => ~friend_of(Henry,Thomas)
rule ~happy(Henry) AND friend_of(Henry,Thomas) => ~happy(Thomas)
```

Notice that, speaking about the rules representing an implication, even if these rules refers to the same elementary statements, the presence of a rule does not affect the results produced when another one is applied. This is because, as told in the previous section, the rules are applied only when a statement is not represented in the description, and only one time in every branch of the inference tree. If `happy(thomas)` , `friend_of(Henry,Thomas)` and `happy(Henry)` are not in the representation, asking the truth value of `happy(Henry)` to the system will make it apply the rule `rule happy(Thomas) AND friend_of(Henry,Thomas) => happy(Henry)` . The other two rules are then used to find the truth values of `happy(thomas)` and `friend_of(Henry,Thomas)` . After that the first rule is no more applicable to search the value of `happy(Henry)` because used previously, the other two rules are applied another time (because it is done in two different branches). The result is a truth value of zero for `happy(Henry)` , as expected just from the first rule. In case that some of the tree truth values are available the research stops earlier, using them directly.

6.4.3 Theoretical notes

Logic is a powerful tool to represent systems. Historically, logic modeling and theorem demonstration represented two important tasks of artificial intelligence research. The system presented in this chapter have some peculiarities that make it different from classical application of logic,like automatic theorem demonstrators. The kind of logic language used in this application shares some characteristics with the ones represented in the following table:

Language	Epistemological Commitment (what is represented)	Ontological Commitment (What is known about)
propositional logic	facts	true/false/unknown
first order logic	facts,objects,relations	true/false/unknown
probability theory	facts	confidence degree$\in [0, 1]$
fuzzy logic	facts with truth degree	known internal values

The syntax used express, about the environment described, facts, objects, relations like first order logic. The ontological commitment does not include true, false or unknown values, but truth degrees[8]. Besides, probability theory is used for gestures. The syntax has some limitations compared to the first order logic:

- the facts declared can only refer to instanced objects;
- the facts declared can only be simple statements;
- the operator OR is not part of the syntax;

The first point, in practice means that the set of facts is actually a set of propositions. Facts like "*all* the screws are lubricated" or "there is *a* rusty bolt". The first sentence should be represented listing the representative sentence for any instanced screw, the second sentence should specify the name of the bolt that is rusty. Also theorem demonstrators transform first order predicates into propositions when performing a demonstration[9], In this case the syntax itself

[8]The truth value 0.5, equidistant from the true (1)and the false(0) is anyway a known value that represent a particular intermediate condition between the two extremes.

[9]The theorem of Herbrand states that any formula implicated by a set of first order logic predicates S, is implicated by a finite subset of proposition inferred by S

forces the user to produce a database of propositions (datalog). The second point is just formal: a conjunction of facts can be represented declaring all of them. The third point creates a bit more difficult problem: in case it represents the cause in a rule, a disjunction can be represented with several rules featuring the same conseguence and the different terms of the disjunction, for example the rule ”if A OR B then C” can be represented as ”if A then C” and ”if B then C”. In case the disjunction is a fact that should be part of the description its representation becomes more tricky: suppose to have the following declarations:

```
function rusty(1)
function lubricated(1)

object bolt
```

A sentence like[10] ”a bolt is lubricated OR rusty” could be defined not by facts but by rules, like:

```
rule ˜lubricated(bolt) => rusty(bolt)
rule ˜rusty(bolt) => lubricated(bolt)
```

These limitations are connected with the major difference between a theorem demonstrator and this system: the former exploits fixed inferential rules to find true sentences from true sentences (inference rules) the latter relies on the rules written by the user to compute the truth values of statements from other statements truth values. The facts are hence limited to a form that makes immediate to retrieve their truth value. Notice that in the previous example the two rules specified explicitly the possibility to infer a fact from a disjunction and the negation of the other term in the disjunction, it is similar to what happens automatically when declaring implications (see 6.4.2)

Another characteristic of the first order logic is that there is no way to demonstrate that a formula is not implied, this can lead to a an infinite loop when trying to do it. In the described system there are two simplifications that prevent the system from being trapped in an infinite loop when computing a truth value:

- relation are represented as N-ary functions
- the same rule cannot be used more than one time in a branch of the demonstration tree (see 6.4.1)

This is because this system is designed to evaluate quickly the truth values of facts, in order to compute the probability of gestures. For the same reason a fact

[10]Again, for the sake of simplicity facts are referred with the meaning that their name suggests, actually they have not any meaning but their truth value.

is considered false when a demonstration is not found.

The facts truth values can change at runtime, new facts can be added and some of them can lead to contrasting results (the fuzzy logic manage it as told in 6.2.5). In this context it represent a change in the state of the system described, because it is supposed to be fully known. In general a logic in which previously taken conclusions can be contradicted by new informations is called *non monotone logic*. These kind of logic can also represent a refinement of knowledge when getting new informations.

Gestures prior probabilities represent the output of this description system (together with truth values) and the *actions* represent a sort of input (together with *notified* facts). Hence, even if prior probabilities are not involved directly in truth values computing, they can influence them through the gesture recognition system, that takes prior probabilities as input and produces action as output. The logical description, together with the gesture recognition system, represent more a complex dynamic system than a proper theorem demonstrator or a database.

6.5 Efficiency and Debugging

This approach to high level modellization brings the advantage of a simple and easy description. An adaptive model like HMM, although more flexible, would be difficult to understand and debug. This solution is efficient when the designer of the system knows well how to describe the environment. An unappropriate logic environment would make the behaivour of the virtual system unrealistic. This is the thing that should be checked during the debug phase. It is interesting to point that a *wrong* prior probability vector or fact would produce a *bug* just when a particular situation occours. This *locality* of effects occours because prior probabilities are computed by active facts. This would make easy to find the error that produces the wrong system behaivour.

6.6 An Example

In this section will be presented, as example, the description of the virtual environment used in the demostrative application described in 8.1.

6.6.1 The Environment

The application consists in the assembling of a simple object: a small box with a removable corck that can be fixed to it by four screws. The environment features the components to build the object (a box, a flat base that is a corck for the box and

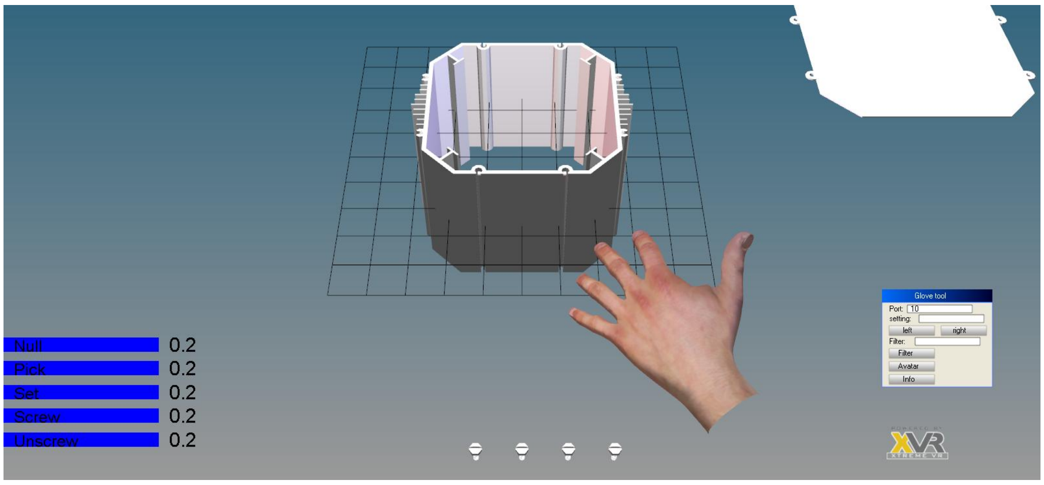


Figure 6.4: The virtual environment represented with XVR

four screws) and an hand controlled by the user. There are also invisible objects called ”holes” representing the female screw treads on the box. To perform the assembling task the operations should be performed in the right order, for example to place the cork no screw should be screwed, and at least a screw should be placed to fix it.

6.6.2 The Code

Here is the code describing the logical environment:

```
function screwed_on(2)
function set_on(2)
function near(2)
function set_on(base1,box1)
function dummy(1)
function inhand(1)
function blocked_on(2)

object hand

hand lefthand[11]

object screw
```

[11]Having the just left hand sensorized allows the developer to use the right hand to program while testing and debugging.

```
screw screw1
screw screw2
screw screw3
screw screw4

object box

box box1

object base

base base1

object hole

hole hole1
hole hole2
hole hole3
hole hole4

fact dummy(lefthand) 1.0

rule screwed_on(screw #1,hole #2) AND set_on(base1,box1) =>
blocked_on(base1,box1)

action null:  dummy(lefthand) => dummy(box1)

action picking:  near(lefthand,screw #1) AND set_on(screw
#1,hole #2) => ~set_on(screw #1,hole #2)
:  near(lefthand,ANY #2) => inhand(ANY #2)
:  near(lefthand,ANY #2) AND inhand(ANY #2) => ~inhand(ANY #2)

action setting:  near(lefthand,screw #1) AND near(screw #1,hole
#2) => set_on(screw #1,hole #2)

action screwing:  near(lefthand,screw #1) AND set_on(screw
#1,hole #2) => screwed_on(screw #1,hole #2)

action unscrewing:  near(lefthand,screw #1) AND screwed_on(screw
#1,hole #2) => ~screwed_on(screw #1,hole #2)
```

```
priors dummy(lefthand)
null 0.2
picking 0.2
setting 0.2
screwing 0.2
unscrewing 0.2

priors inhand(screw)
null 0.3
setting 0.4
picking 0.3

priors inhand(base1)
null 0.3
setting 0.4
picking 0.3

priors near(lefthand,screw #1) AND ~inhand(screw #1) AND
~screwed_on(screw #1,hole #2)
null 0.4
screwing 0.3
picking 0.3

priors near(lefthand,screw #1) AND ~inhand(screw #1) AND
screwed_on(screw #1,hole #2)
null 0.5
unscrewing 0.5

priors near(lefthand,base1) AND ~inhand(base1) AND
~blocked_on(base1,box1)
null 0.5
picking 0.5
```

6.6.3 The set of actions

The possible action for the user are:

- *null*: this action consist in doing nothing. It is represented as an actual gesture in order to be actively discriminated from the other gestures;

- *pick*: this action consist in picking up an object when the hand is empty and in relasing it when hold. As explained in 4.3 the data stream produced by the gestures is transformed through a *fast fourier transform*. This makes the gestures of closing and opening the hand to pick and relase the objects to be recognized as the same gesture (they are also presented together in the training set). After picking an object, while the hand is kept closed, the system recognizes a *null* gesture, when the hand is opened the pick gesture is recognized again and the object is relased;

- *set*: this action sets the hold object in a position suitable for assembling it on the nearest object (if it is possible). It places the screws on the screw treads and the corck on the box;

- *screw*: this action consist in screwing a screw in a tread of the box;

- *unscrew*: this action consist in unscrewing a screw from the box;

6.6.4 Running The Application

The 3D environment is implemented with XVR. Since the logical description works just with fuzzy values the states of the objects are *fuzzyfied* by the XVR script itself.

When the application starts all the condition specified through the command `Priors` have truth value 0. Just the *always true* condition `dummy(lefthand)` produces the vector [0.2 0.2 0.2 0.2 0.2] that represent an equal prior probability for every gesture. Once the hand approaches the corck, the function `near(lefthand,base1)` grows linearly with the negated distance. The prior probabilities are adjusted according to

```
priors near(lefthand,base1) AND ~inhand(base1) AND
~blocked_on(base1,box1)
null 0.5
picking 0.5
```

The corck `base1` is than picked up, the XVR script *notifies* to the logical description the fact

```
inhand(base1) 1.0
```

that makes the exposed condition false (= 0). The prior probabilities vector

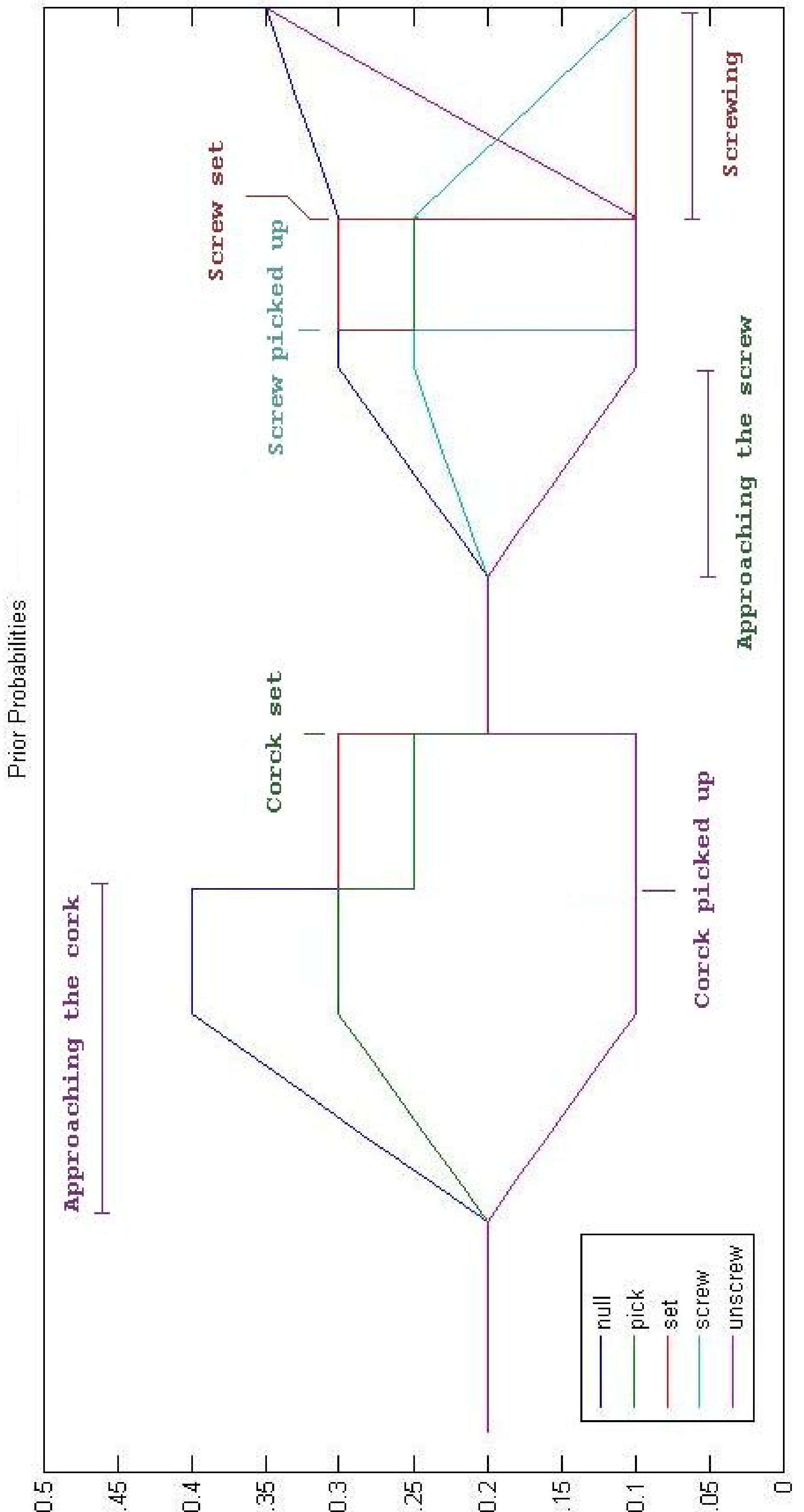
```
priors inhand(base1)
null 0.3
```

```
setting 0.4
picking 0.3
```

enters suddently in the computing of the actual probability vector with weight 1. Setting the corck on the top of the box is the more likely acttion to be performed. Also relasing it with a pick is a possible way to use it. Once the corck is set on the box the screws should be used to block it. Approaching a screw with the hand triggers a condition similar to the one seen for the cork:

```
priors near(lefthand,screw #1) AND ~inhand(screw #1) AND
~screwed_on(screw #1,hole #2) AND ~inhand(base1)
null 0.4
screwing 0.3
picking 0.3
```

That holds until the screw is picked up or screwed. the screw is than to be brought on a hole. set and screwed. It should be done for all the four screws. The changes of the probability vector during the operation are represented in figure 6.5.


Prior Probabilities
Approaching the cork
Corck picked up
Corck set
Approaching the screw
Screw picked up
Screw set
Screwing
null
pick
set
screw
unscrew
0.5
.45
0.4
.35
0.3
.25
0.2
.15
0.1
.05
0

Chapter 7

The Hardware

The capturing system used consist in:

- a sensorized glove, built at PERCRO laboratories;
- an electromagnetic tracker of position and orientation Polhemus Liberty^{TM};
- a visual tracking system Vicon^{TM};

The glove and the Pohlemus tracker are used together for the application in virtual environment: they allow to track the position of the hand and the fingers. The Vicon tracking system is used to produce a large set of movements to test the scalability of the proposed system.

7.1 The PERCRO Dataglove

The dataglove[1] is sensorized with 11 hall effect potentiometers. Sensors geometry and position is designed to retrieve angular displacement. This guarantees a measurement that is in practice independent from the length of users fingers. Of the 11 sensors 8 are put, two per finger on index, middle, ring and little. The other 3 are on the thumb. The movement perceived by the glove are just rotations, abductions and adductions have not dedicated sensors. A 16 bit unsigned integer sample is produced for every sensor at every sample time. Actually the transmitted value is the sum of 16 consecutive samples, so the glove onboard electronics embeds a low-pass filtering. The glove sends data to a receiver via bluetooth link. The available bandwidth is of 115,200 bps, that allows a flow of 492 frame packets per second. Usually the used band is no larger of 100 packets per second: 100 Hz is the speed guaranteed by specification.

[1]This device is described in [OPR]

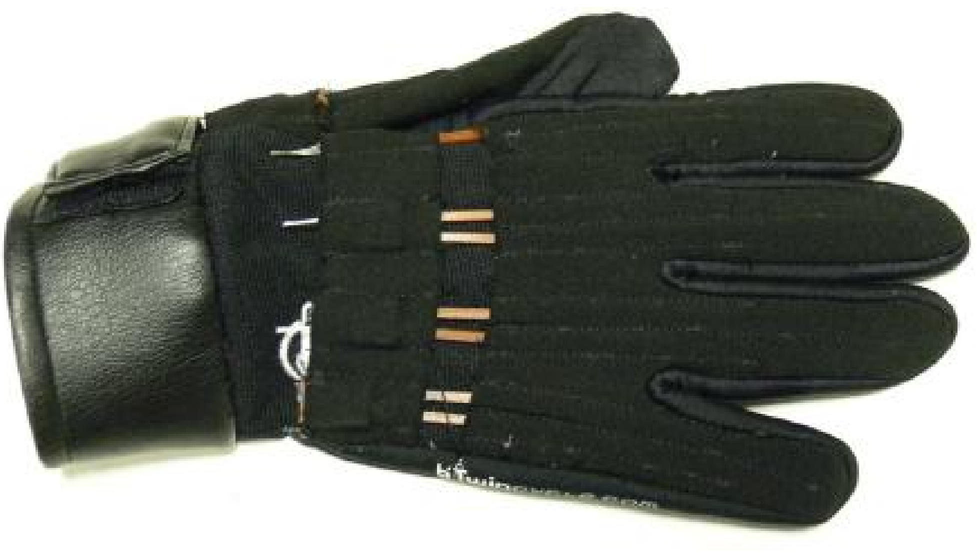


Figure 7.1: PERCRO Dataglove

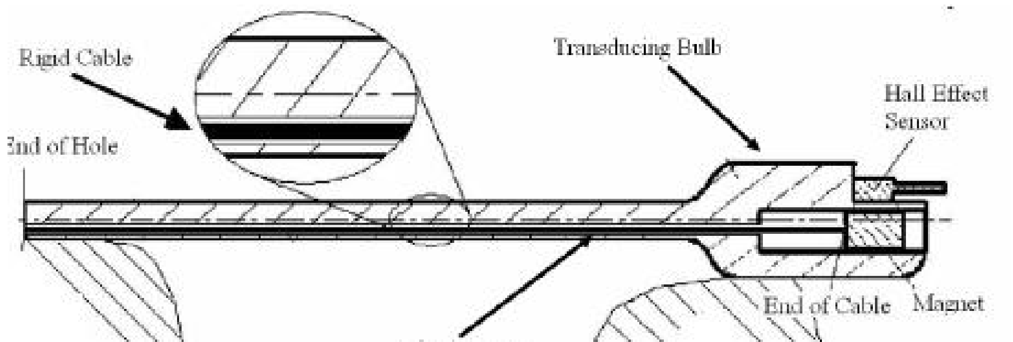


Figure 7.2: Goniometric sensor

7.2 The Pohlemus tracker

The Pohlemus Liberty tracker[2] is an electromagnetic system capable of tracking up to four sensors. It detects position and orientation, allowing to track the sensor on six degrees of freedom. It can work to 240 Hz and it is connected to the computer through USB or RS232. The resolution is of 0.0004 cm in position and of 0.0012 degrees in orientation.

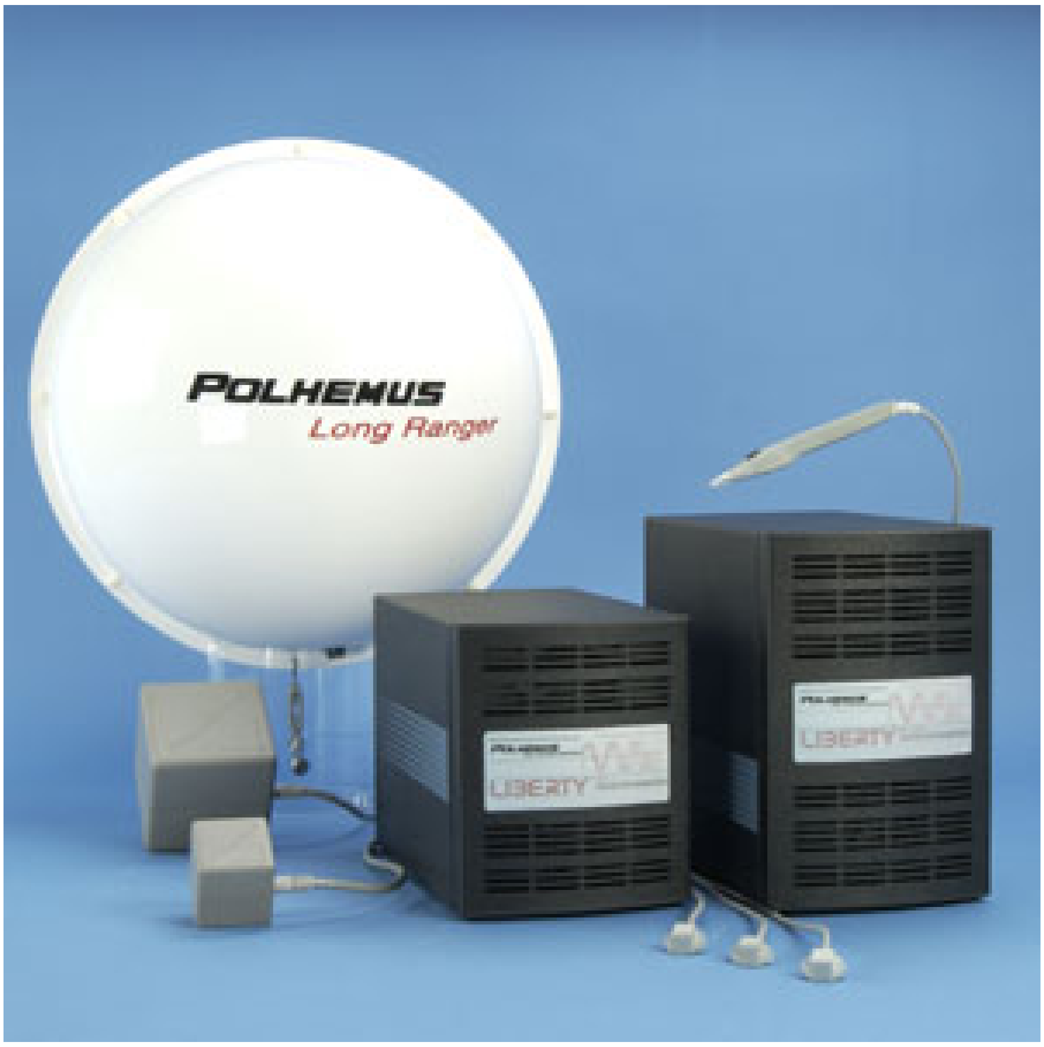


Figure 7.3: Pohlemus Liberty hardware.

[2]Pictures and informations in this sections have been retrieved on www.pohlemus.com

7.3 The Vicon Tracker

The Vicon is an optical tracking device. It is composed by a set of cameras, a dedicated computing hardware and an elaboration software. This system can locate the position of visual markers and, using a model of the tracked object dynamic, reconstruct its cinematic. The main advantages are the precision (13 millions pixel per camera), the speed (up to 2000 fps) and the total freedom of movements that allow the tracked user to perform natural movements. The main limitation is that it is an expensive system and it requires a non trivial set up. This mean that solutions based on it are not suitable to be exported from this experimental context.

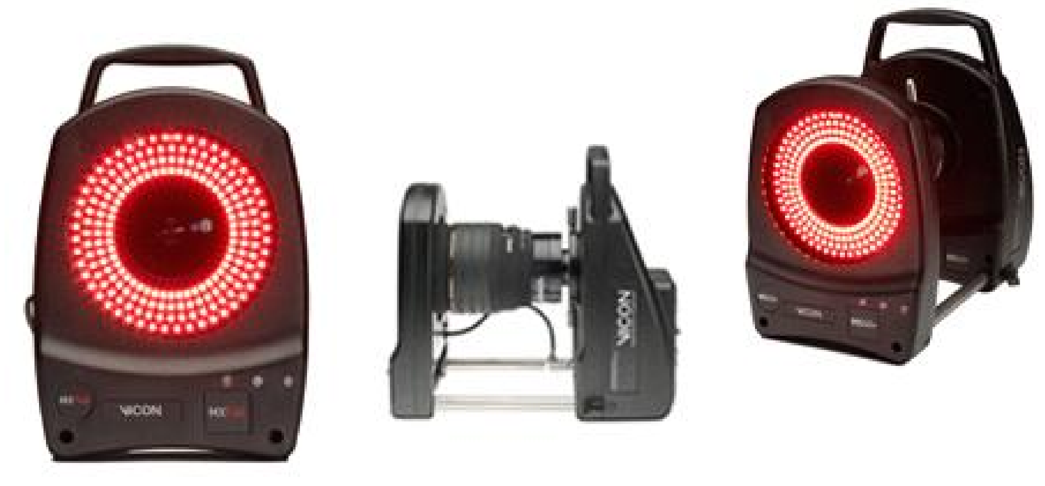


Figure 7.4: Vicon camera

Chapter 8

Demonstrative Applications

8.1 Assembling Application

This application consists in assembling a small computer case in the virtual environment. The user interacts with the components through an avatar of his/her hand. The recognized gestures trigger effects in the virtual environment. The tridimensional representation of the environment is updated coherently with the logical description. In order to build the case components should be put in place, set with the right orientation and than blocked. To complete the task the operations should be performed in the right order. The case is made with a box, a base and four screws to fix them together. The recognized gestures are:

- null;
- pick;
- set;
- screw;
- unscrew;

The gesture *null* does not produce any action, it is recognized when the hand is almost still. It is needed to avoid recognition of random gestures when the use is not doing something. The gesture *pick* is used to take and release objects. There is not a dedicated gesture ”release” for the sake of simplicity: the training set is produced from a series of alternated hand opening and closing, producing a training set just with the movement of opening and another one just with the movement of closing would have required a more complex procedure than just recording it. *Set* is used to put an object in the right position for assembling. This overcomes the fact that in this application there is not a force feedback and the

2008-02-13 16-37-39-68.JPG

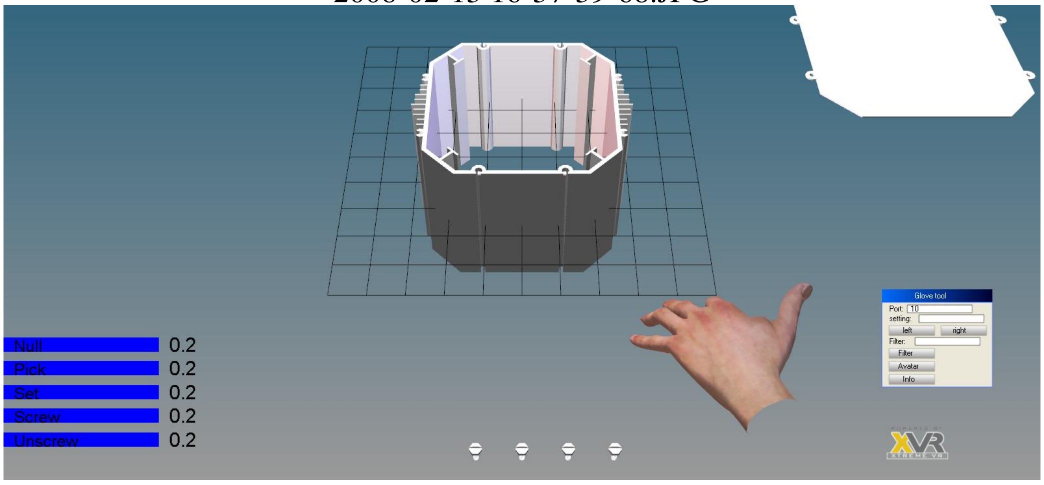


Figure 8.1: The virtual environment at the beginning of the task.

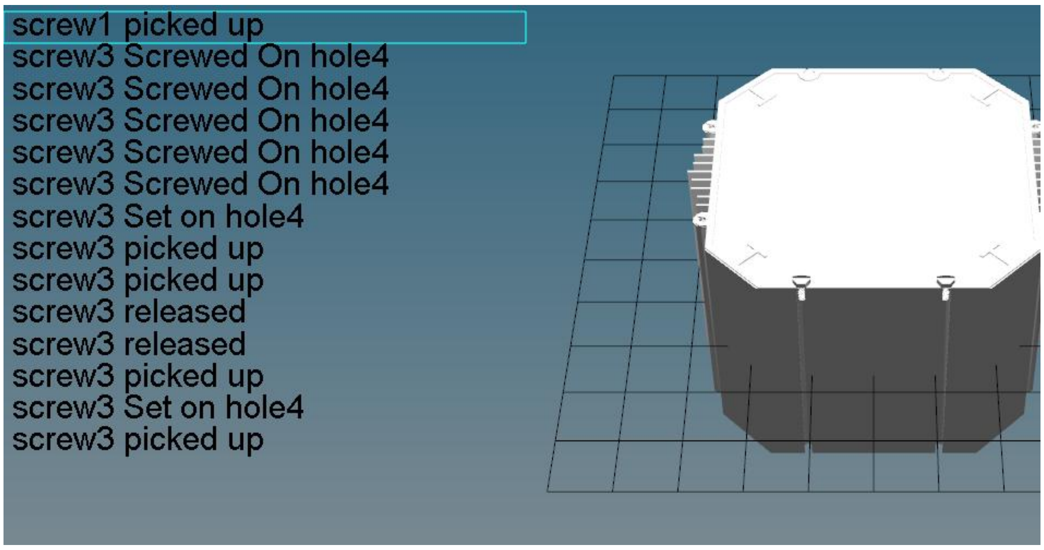
right position is hence difficult to find. Notice that ”being set” is a state of an object represented in the logical environment and it is anyway needed to mount an piece. The gestures *screw* and *unscrew* are used to fix screws on the box and to remove them. There are 5 degrees of screwing represented by a natural number, when the gesture *screw* is recognized the number is increased, on the opposite unscrewing decreases the number.

8.1.1 Video Messages

There are some informations displayed on the tridimensional scene: a list of the action performed and a graph representing prior probabilities for the five possible gestures. The list works both as a check of the recognized gestures and of a record of the steps needed to perform the task. As told in the introduction (chapter 1) one of the capabilities of a gesture recognition system is to build automatically a manual for the simulated procedure. The graph of prior probabilities is put mainly for demonstrative purpose. It makes possible to appreciate how they changes with the evolution of environment states. Besides this graphic can be useful also in a real application to verify the coherence of the description with the tasks and, in case improve it.

8.1.2 Complete Logic Description

The logic description used in this application is a bit more complex than the one presented in 6.6.2 where the accuracy was less important than presenting a lean

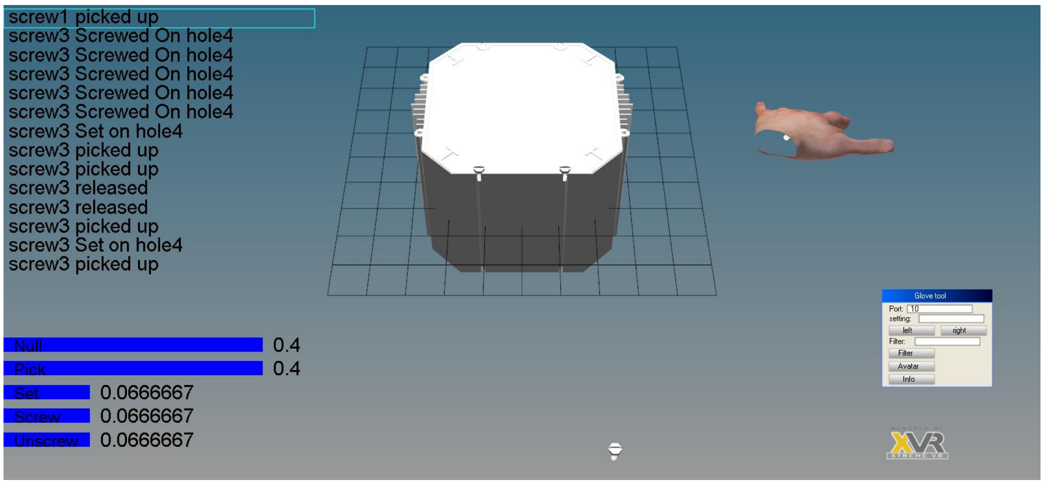


```
screw1 picked up
screw3 Screwed On hole4
screw3 Screwed On hole4
screw3 Screwed On hole4
screw3 Screwed On hole4
screw3 Screwed On hole4
screw3 Set on hole4
screw3 picked up
screw3 picked up
screw3 released
screw3 released
screw3 picked up
screw3 Set on hole4
screw3 picked up
```

Figure 8.2: List of performed actions

and clear example.
Here is the code:

```
function screwed_on(2)
function set_on(2)
function near(2)
function set_on(base1,box1)
function dummy(1)
function inhand(1)
function blocked_on(2)

object hand

hand lefthand

object screw

screw screw1
screw screw2
screw screw3
screw screw4

object box
```



```
screw1 picked up
screw3 Screwed On hole4
screw3 Screwed On hole4
screw3 Screwed On hole4
screw3 Screwed On hole4
screw3 Screwed On hole4
screw3 Set on hole4
screw3 picked up
screw3 picked up
screw3 released
screw3 released
screw3 picked up
screw3 Set on hole4
screw3 picked up
```

```
Null      0.4
Pick      0.4
Set       0.0666667
Screw     0.0666667
Unscrew   0.0666667
```

Figure 8.3: Prior probabilities influenced by the context

```
box box1

object base

base base1

object hole

hole hole1
hole hole2
hole hole3
hole hole4

fact dummy(lefthand) 0.7

rule screwed_on(screw #1,hole #2) AND set_on(base1,box1) =>
blocked_on(base1,box1)

   action null:  dummy(lefthand) => dummy(box1)

action picking:  near(lefthand,screw #1) AND set_on(screw
#1,hole #2) => ~set_on(screw #1,hole #2)
:  near(lefthand,ANY #2) => inhand(ANY #2)
:  near(lefthand,ANY #2) AND inhand(ANY #2) => ~inhand(ANY #2)
```

```
action setting:  near(lefthand,screw #1) AND near(screw #1,hole
#2) => set_on(screw #1,hole #2)

action screwing:  near(lefthand,screw #1) AND set_on(screw
#1,hole #2) => screwed_on(screw #1,hole #2)

action unscrewing:  near(lefthand,screw #1) AND screwed_on(screw
#1,hole #2) => ~screwed_on(screw #1,hole #2)

priors dummy(lefthand)
null 0.2
picking 0.2
setting 0.2
screwing 0.2
unscrewing 0.2

priors inhand(screw) AND ~near(lefthand,hole)
null 0.5
picking 0.5

priors inhand(screw) AND near(lefthand,hole)
null 0.3
picking 0.2
setting 0.5

priors inhand(base1) AND near(lefthand,box1)
null 0.5
setting 0.5

priors inhand(base1) AND ~near(lefthand,box1)
null 0.5
picking 0.5

priors near(lefthand,screw #1) AND ~inhand(screw #1)
AND ~screwed_on(screw #1,hole #2) AND ~inhand(base1) AND
~set_on(screw #1,hole #2)
null 0.4
picking 0.6

priors near(lefthand,screw #1) AND ~inhand(screw #1)
AND ~screwed_on(screw #1,hole #2) AND ~inhand(base1) AND
```

```
set_on(screw #1,hole #2)
null 0.3
picking 0.2
screwing 0.5

priors near(lefthand,screw #1) AND ~inhand(screw #1) AND
screwed_on(screw #1,hole #2) AND ~inhand(base1)
null 0.5
unscrewing 0.5

priors near(lefthand,base1) AND ~inhand(base1) AND
~blocked_on(base1,box1)
null 0.4
picking 0.6
```

Appendix A

Further Implementation Details

This Appendix is a brief summary on the technical solution used to implement the application described in the previous chapters. The code and the Matlab/Simulink schemes are partially reproduced when needed for the explanation. It is anyway not intended as an exaustive exposition of the source code.

A.1 Application Overview

In order to work the recognition system needs the following elements:

- data sampling;
- data encoding;
- sequence recognition system;
- logical description library;
- interactive application;

During the execution a Matlab/Simulink scheme (recalling several scripts) and the XVR interactive application are executed simultaneously, they exchange data through UDP protocol. Data sampling is performed integrating an acquisition library in Matlab for the finger position, and another library in the XVR application for the position tracking: the first has been developed at PERCRO labs for the glove, the second comes together with Polhemus tracking system. The encoding (frequency analysis, PCA and neural quantization) are perfromed directly in Matlab, that allows a quick access to the tools needed. The sequence recognition system and the logical description library are compiled as *dll*[1] and recalled the former by Matlab scripts, the latter by the XVR application.

[1]The application has been developed under windows

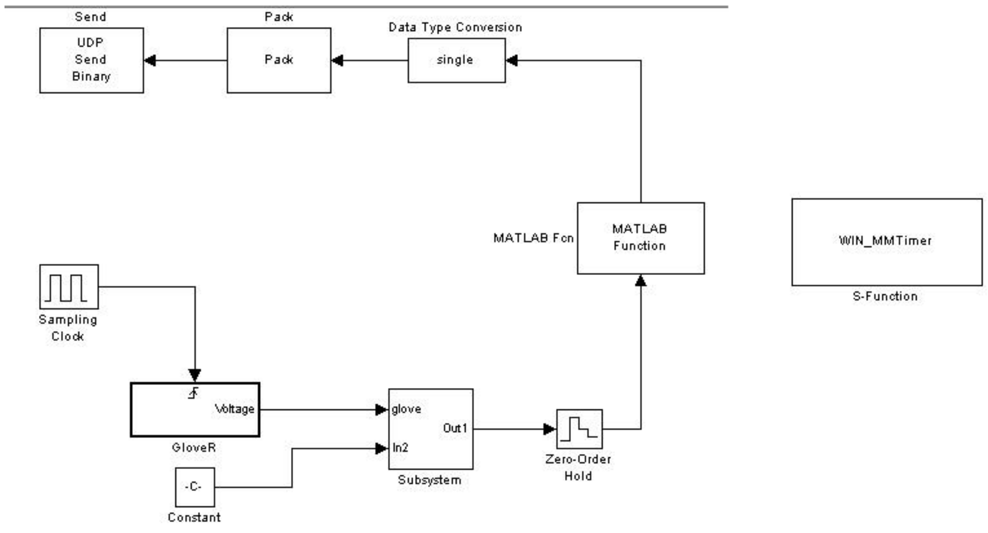


Figure A.1: Simulink Scheme

The dll implementing the sequence recognition uses the functions shown in A.2 based on the principles explained in chapter 5. These functions are thought to work on sequences, hence the dll incorporates also global variables and special functions to buffer samples received from Matlab[2], build up a sequence of them and than classify the sequence obtained. The classification function is called at every sampling time: if the buffer is not full the sample is added, if the buffer is ready the sequence is classified and the buffer is flushed and the process starts again. An integer number representing the gesture recognized is the returned value; it is kept unchanged whil filling the buffer, during the first recognition[3] it is set to zero, value that represent the *null gesture*.

The figure A.1 represents a Similink scheme incorporating the data acquisition, the sequence recognition and the comunication to XVR through UDP protocol. The block labeled ”S-Function” is needed to syncronize the scheme with the sampling[4];the three upper blocks are used to communicate via UDP to XVR; the ”GloveL” block emits the data sampled from the glove; the other blocks are added just to manage matters of signal routing and syncronization between blocks.

The block labeled ”Matlab Fcn” incorporates encoding and sequence recognition, at every sample step the following script is executed[5]: `function`

[2]Exchanging complex data structure is complicated and so it is better to let the library built the sequence that actually is a *vector of vectors*

[3]This phase lasts for less than a second. If the buffer is too big the recognition could become too slow to be applied

[4]It exploits code developed at PERCRO labs

[5]The sintax is the one typical of matlab scripts

```
[y]=classification(u)
global COEFF; global net1 net2 net3;
global PrC
global IDGESTURE
PrC=[PrC;u(1:11)'*COEFF];
if size(PrC,1)>=50;
PRIORS=single(u(12:16));
Sample1=fft(PrC(1:50,1));
Sample2=fft(PrC(1:50,2));
Sample3=fft(PrC(1:50,3));
serie1=vec2ind(sim(net1,Sample1(2:25,:)));
serie2=vec2ind(sim(net2,Sample2(2:25,:)));
serie3=vec2ind(sim(net3,Sample3(2:25,:)));
symbol=int32([serie1;serie2;serie3]);
psymbol=libpointer('int32Ptr',symbol);
pPRIORS=libpointer('singlePtr',PRIORS);
PrC=PrC(7:50,:);
[IDGESTURE]=calllib('Classifier','Classify',pPRIORS,psymbol,8);
end
end
y=IDGESTURE;
end
```

This is a function, and it is called by Simulink during the execution. The keyword `Global` global variables are imported, they represent:

- `COEFF`: a matrix of coefficients to transform the sample series in their principal components;
- `ne1,net2,net3`: neural network implementing the codebook;
- `PrC`: an array that accumulates samples for a given time window.
- `IDGESTURE`: the last gesture identified;

Global variables have a *static* time of life, this means that they hold their value between two calls to the function. This makes possible to build up the `PrC`to compute the PCA and the fourier transform on it, as well as keeping structures prepared during the set up in the other global variables. Every seven samples received from the glove producing a symbol from the sample in PcR, that is 50 samples long (the time windows are overlapping) `callib` evokes the dll function to classify it.

A.2 Sequence Recognition library

Sequences classification is performed by a library written in C++. Methods and structure implemented allows to create, train, save, load and use the Classifier.

A.2.1 Methods and Structures

The library feature these structures[6] with related methods:

- **sequence** - Implements a list of sequences
- **MxMiHMM** - Describes an HMM and features methods to adapt it during training and to compute probabilities needed for classification
- **Classifier** - Describes a usable classification system
- **Trainer** - Derived from Classifier [7] this structure includes the training set. It features methods to train the classifier and save it once trained.

A.2.2 Detailed Code

Structure ”sequence”

Definition:

```
struct sequence
{
double* prob;
double Tprob;
sequence* next;
int** sample;
int lenght;
int model;
}
```

The structure is a simple implementation of a list through a the pointer `next` that points to the next object. The field `sample is a pointer to the sequence (an array of arrays of integer), the field length` represents the number of samples featured and the field `model` represents the model

[6]”Classes” is a more precise term for C++ structures defined by internal field and a set of functions (and other features typical of C++ class implementation). Sometimes the term can be confusing in this context where ”classes of objects are given”

[7]it means that a ”Trainer” object is also a ”Classifier” object, but not vice versa. ”Trainer” has all the methods and the fields of ”Classifier”

supposed to be recognized as the most probable emitter for the sequences (that is the label of the target class). The field `prob` points to the vector of probabilities $P(S|M_i)$ for every class i, Tprob is the total probability $P(S)$. Both `Tprob` and `prob` are updated at every step during the training algorithm, and they are not proper data of the sample sequence. The list implemented by `sequence` is used by the trainer to hold the training set. When classifying, the system can take raw arrays of symbols as input.

structure ”MxMiHMM”

Definition:

```
class MxMiHMM
{
int Nstat;
int Nemis;
int Nobs;
struct parameters

double* A;
double* B;
double* pi;
;
double* A;
double* B;
double* pi;
void sumP(parameters* dest,parameters* ad);
void AddGrad(parameters* ad);
void divP(parameters* dest,double p);
void Projection(parameters* p);
void ProjectionBorderline(parameters* p, parameters*
OnConstraint);
int MIndex(int row,int col);
void Normalize();
void Gradient(int* seq,int seqlen,parameters* ret);
void initparam(parameters* p);
void deleteparam(parameters* p);

public:
MxMiHMM(const MxMiHMM&);
MxMiHMM& operator=(const MxMiHMM&);
MxMiHMM(int Ns,int Nem);
void Run(int* out,int len);
MxMiHMM();
MxMiHMM();
void trainstep(const sequence* sampleseq, const sequence*
negseq, double* Pmodels, double epsilon);
void Forward(double** alpha, int* sample,int seqlen, int
time);
void Backward(double** beta, int* sample,int seqlen,int
time)
double ForwBack(int* sample, int seqlen);

friend ostream& operator<<(ostream& to, const MxMiHMM model);
friend istream& operator>>(istream& from, const MxMiHMM
model);

};
```

The fields of a `MxMiHMM`represent the number of hidden states (`Nstat`), the number of possible values for every observable (`Nemis`), the number of observables (`Nobs`) and the matrices A, B and π (`A`,`B`and `pi`respectively). The structure `parameters` is used to represent model parameters involved in algorithm operations. Both for the `MxMiHMM`itself and `parameters` A and B are represented row by row as simple arrays: the function `MIndex()` is used to indicize the values in the array given the index in the matrices. `Forward()` and `Backward` compute reapectively the functions $\alpha(t)$ and $\beta(t)$, `ForwBack()` computes the emission probability for a sequence, `Run()` produces a random sequence emitted by the model. Streaming operators are used when loading or saving a classifier. Data are not formatted in a particular way: the arrays are just saved as stored in memory. The method `Normalize` normalizes the parameters, it is used when creating the model to produce a coherent initialization, and during the training to prevent numerical approximations to destroy data coherency.

Structure ”Classifier”

Definition:

```
class Classifier
{
protected:
MxMiHMM* Models;
int NModels;
public:
Classifier()
int Classify(int **seq, int len);
int LoadClassifier(char* nomefile);
int SaveClassifier (char* nomefile);
};
```

The class, representing the sequence classifier, includes a set of HMMs, contained in the vector pointed by `Models` . The function `Classify()` exploits the principles exposed in 5.2 to classify sequences. `LoadClassifier()`and `SaveClassifier()` load and save the classifier. This class does not provide methods to modify the methods, `SaveClassifier()` is hence used by the derived class `Trainer` in order to save the classifier once it is trained.

Structure ”Trainer”

```
class Trainer :  public Classifier

sequence* Samples;
int NSamples;
public:
Trainer();
Trainer();
double Train(int steps,double epsilon,double*
ModelProb=NULL);
int AddSequence(int** sample, int len ,int model);
int AddModel(MxMiHMM& ToAdd);
int AddModel(int stats,int obs);
int LoadTrainer(char* nomefile);
int SaveTrainer(char* nomefile);
;
```

The class `Trainer` , derived from `Classifier` , allows to create and train a classifier. The training set is hold by the pointer `Samples` that works as base for the list of `sequence` objects. Besides the inherited functions `LoadClassifier()`and `SaveClassifier()`, `Trainer` has its own load/save methods, `LoadTrainer()`, and `SaveTrainer()`, that save and load also the training set. Models could be added to the classifier using the method `Addmodel()` (that, by overloading, can accept an existing HMM or the paremeters to create a new one). The method `Train()` implemets the training algorithm itself. The parameter `epsilon` is the inverse of μ and is used to tune convergence speed, `ModelProb` is the vector of prior probabilities P(M). The value returned by `Train()` is the goal function 5.8

A.3 Logical Environment management library

The virtual environment description is implemented by a library written in C++. The library features fuctions to load a description from a file, to get information from it and interact with it changing its logical states.

A.3.1 Methods and Structures

The main structure implementing the logical description is **StateManager** that represent all the informations avaiable about the environment. To represent the various part of the description (see 6) several structures are defined. These structures are thought to be referred by **StateManager** and not to be referred directly. The user can specify facts, rules actions etc. just using strings at runtime.

A.3.2 Detailed code

Structure "StateManager"

Definition:

```
class StateManager
{
//tables
vector<function> Functions;
vector<object> Objects;
vector<objecttype> Objecttypes;
vector<rule> Rules;
vector<action> Actions;
vector<priors> Matrix;
list<fact> Facts;
//Utilities
int unique;
vector<int> AnyIndex;
int str2num(string str);
bool readfact(istream& from,list<fact> &factls,bool untold);
void readrule(istream& from,vector<rule> & rulelist);
void instantiate(list<list<fact> > &real_inst);
void rule_instantiate(list<fact>&left,fact& right_tpl,fact&
right_inst);
void notify(fact& newfact,bool temporary);
double evaluate(list<fact>& terms,vector<int> exploited);
double evaluate(fact &term, vector<int> exploited);
public:
StateManager(char* file);
vector<double> GetPriors();
void notify(string f,bool provvisory); // notify a fact
void flushtemporary();
double evaluate(string f);
double revaluate(string f);

   friend ostream& operator<<(ostream& to,StateManager& S);

   bool act(string actn);
};
```

This structure holds all the data describing the environment: functions, objects and their types, rules and implications, actions and prior probabilities for gestures. All of them but implications are represented by the omonimous structures, implication are stored in memory as a set of rules. The public methods are

- `StateManager`: The class constructor, parses a text file with the environment logical description description updating the fields.
- `notify`: to notify facts, making them part of the logical description or updating their truth value. This function has an overloaded private method that handles the fact in the form in which it is represented in memory (using the structure `fact`) that is called once the string has been parsed;
- `evaluate`and `revaluate`: to get the truth value of a fact. Using `revaluate`the fact is not searched in the description but computed, through the rules, from the other facts, and memorized.In case the fact already exist in memory its truth value is replaced with the computed one. Also in this case there are private overloaded methods. one of them evaluates the single term truth value, the other the value of the left side of a rule (logical product of terms). They are used recursively to implement the inference algorithm seen in 6.4;
- `flushtemporary`: to delete every fact flagged as provvisonal;
- `operator<<`: to print the descripiton, useful during the debug and the appication set up;

The private method `rule_instantiate` specializes a rule to instanced object to compute it's conseguences.

Data Structures

The representation exploit several structures to hold data in memory:

- Objecttypes: types declared with the keyword ”object”, used in parsing rules. they have a name and an identifier number
- Objects: declared objects, with name, type and identifier number;
- fact: a fact (function applied to an object) with truth value;
- Rules: rules to infer facts from facts. they hold a list of facts representing causes and a fact representing the effect. A set of rule can represent an implication.
- Actions: actions the user can perform. they hold some facts as effects;
- Priors: rules to compute prior probabilities for every action;
- Functions: hold the function name and a relative number, used to parse facts in form of strings;

The identifier numbers for functions and objects are used to represent facts in a lean and quickly processable form. Follows the C++ declaration of these structures:

```
class function
{
public :
string name;
int arity;
function();
function(string n,int a) {arity=a; name=n;}
bool check(string n,int a) {return ((a==arity) &&
(n==name));}
function& operator=(function& f);
};

class object
{ public :
string name;
object()
object(string n) name=n;
bool check(char* n) return (n==name);
};

   class fact
{ public :
int funcnum;
vector<int > objects;
vector<int > instances;
double truth;
bool temporary;

bool check_tpl(fact& test,double& t,vector<objecttype>
&Objecttypes);
void specify(int inst,int ty,int ob);
bool check(fact& test,double& t);
};

   class priors {
public :
list<fact> trigger;
vector<int > actindexes;
vector<double> prob;
priors(list<fact>& tr,vector<int > &ai,vector<double>
&pr){trigger=tr;actindexes=ai;prob=pr;}
};

   class rule

public :
vector<int > variables;
fact rightterm;
list<fact> leftterms;
rule(list<fact> left,fact right);{rightterm=right;leftterms=left;}
rule(){};
};

   class action
{
public :
string name;
vector<rule> tasks;
unsigned int mode;
```

```
action(const vector<rule> &t,string nam) {name=nam;tasks=t;}
};
```

Appendix B

Hidden Markov Models

This appendix explains what is an hidden markov model with the notation used in the previous chapters[1]. It is to point that several kinds of model are referred as HMM in different context, here the model used in the implemented application will be described. Other variants, mentioned in the state of art chapter 2, will not be explained in details. A basic article on HMM is [LR86].

B.1 Markov Chains

A Marcov chain is the basis of an HMM. It is a discrete-time random process where the next state depends only on the present state and does not directly depend on the previous states. This is known as *Markov property*[2] and could be formally expressed as:

$$P(X_n|X_{n-1} = x_{n-1}, X_{n-2} = x_{n-2} \dots X_1 = x_1) = P(X_n|X_{n-1}) \tag{B.1}$$

The possible values for X_i are discrete and finite.

The changes of state are called transitions and are characterized by the *transition probability*

$$a_{ij} = P(X_n = j|X_{n-1} = i) \tag{B.2}$$

where i and j are natural numbers assumed as labels for the states. Transition

[1]Different articles/books use different notation to describe hidden markov models, here a notation inherited from speech recognition literature is used.

[2]The concept is generalized as *markovian process of n^{th} order* where n is the number of previous states to be taken in account. the *Markov chain* here described is hence a *markovian process of first order*

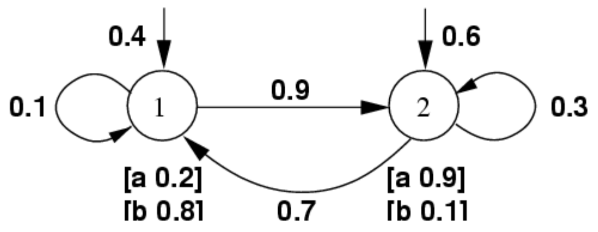


Figure B.1: Graphic representation of a 2-state HMM

probabilities can be collected in a matrix called the *transition matrix*

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \\ & & \ddots & \\ & & & A_{nn} \end{bmatrix} \tag{B.3}$$

To fully characterize a Markov chain it is also needed to specify the probability for the process to start in a given state

$$\pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \vdots \\ \pi_n \end{bmatrix} \tag{B.4}$$

the probability for the process C to produce a certain sequence Z of states z_t, T samples long, is

$$P(Z|C) = P(X_0 = z_0) \prod_{t=1}^{T} P(X_t = z_t | X_{t-1} = z_{t-1}) \tag{B.5}$$

for every sample time t a vector π_t can be defined

$$\pi_t = A^t \pi \tag{B.6}$$

whose componets are the probabilities of being in a certain state at the time step t.

B.2 Hidden States and Emission Matrix

An HMM consists in a Marcov process not directly observable, being the states hidden. The process is observable only through emitted symbols. For every state

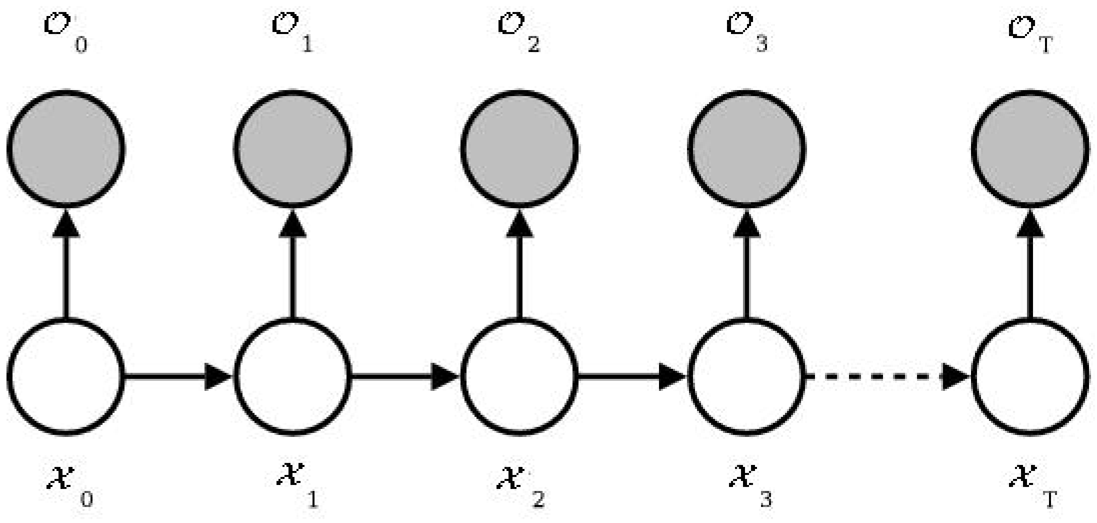


Figure B.2: A ladder diagram for HMM

an emission probability of every possible symbol should be specified. The probabilities can be collected in an emission matrix usually called B;

$$B = \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix} \tag{B.7}$$

where

$$b_i = \begin{bmatrix} P(o_1|X = x_i) \\ P(o_2|X = x_i) \\ \vdots \\ P(o_n|X = x_i) \end{bmatrix} \tag{B.8}$$

where $o_1, o_2 \ldots o_n$ are the n observable symbols and $P(o_k|X = x_i)$ is the probability for the symbol o_k to be emitted when the internal state is x_i.

The figure B.1 is a typical graphic reprensntation of an HMM. The circles represent states and the arcs possible transition with the associated probability. In picture B.2 the HMM is represented unfolded through time: lighter nodes represent the state at a certain time and darker nodes represent the emission. The structure is the same of a bayesian network [Bis06].

B.2.1 Vectorial Output Symbols

A model of HMM with more than one output variable is described in [Bis06]. for every component i of the output vector an emission matrix B_i is specified. The emission of every symbol is independent from the other so that

$$P(\mathbf{o}|X = x_n) = \prod_{k=1}^{N} P(o^k|X = x_n) \tag{B.9}$$

where o^k is the k^{th} component of the symbol $\mathbf{o}$.

B.3 Notation

Here the principal concepts related to HMM, and the way they are referred also in the other chapters, are summarized. The internal states and the emitted symbols are labeled with natural numbers. As told in the previous sections an HMM is characterized by the matrices

- A of transition probabilities, whose components a_{ij} represent the probability of transition from the componet state labeled i to the state labeled j. By definition $\sum_{j=0}^{N} a_{ij} = 1$, where N is the number of possible internal states.
- B of emission probabilities, this can be also an "array of matrices", one for every observable series, whose components b_{ji} represent the probability of emission for the symbol j when the system , is in the state labeled i.By definition $\sum_{j=0}^{M} b_{ji} = 1$, where M is the number of possible internal states.
- π of initial probabilities, where π_i is the probability for the system to start in the state i.

Being θ[3] the vector of all parameters characterizing the HMM [4] the emission probability of a sequence S could be written as $P(S|\theta)$. In this context a different notation is used

$$P(S|M_\theta) \tag{B.10}$$

Where M is the model with parameters θ. Using this notation the role of a cause (whose effects are modelled with M_θ) in producing S as evidence is enphatized. In the context of gesture recognition $P(S|M_\theta)$ can be read as *the probability of observing* θ *when the gesture modelized by* M_θ *is observed*, in other word M is a gesture as sthocastic process, S is its realization. The samples of the sequence S are labeled with a subscript t indexing the time step, so S_t is the observed symbol of the sequence S at time t. So, $P(S_t|M_\theta)$ is the probability of observing the symbol S_t exactly as t^{th} emitted by the model M_θ.

[3]Also λ is commonly used in HMM literature.

[4]In general the parameters are the entries of A,B and π matrices. Sometimes, when there are some additional constraints on the model, the parameters could be a different set

It is also useful to define functions that are referred in algorithms:

$$\alpha_t(i) = P(S_1, S_2, \ldots, S_t, X_t = x_i | M_\theta) \tag{B.11}$$

that is the probability of the sequence S to be observed until the time t and of the internal state to be x_i, it is called *forward probability*. It is computed efficently with an iterative algorithm:

$$\begin{array}{ll} \alpha_1(i) = \pi_1 b_i(S_1) & for\, 1 \leq i \geq N \\ \alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(j) a_{ij}\right] b_j(S_{t+1}) & for\, t = 1,2..T,\ 1 \leq j \geq N \end{array} \tag{B.12}$$

where N is the number of states and T is the number of samples in S. This algorithm has complexity $N \cdot T$[5]. In a simalar way a *backward variable* β can be defined as:

$$\beta_t(i) = P(S_{t+1}, S_{t+2}, \ldots, S_T, X_t = x_i | M_\theta) \tag{B.13}$$

that is the probability of the sequence S to be observed starting from the time t until the end, and of the internal state to be x_i. It is called *Backward probability*. As for α it is computed with an iterative algorithm:

$$\begin{array}{ll} \beta_T(i) = 1 & for\, 1 \leq i \geq N \\ \beta_t(j) = \sum_{j=1}^{N} a_{ij} b_j(S_{t+1}) \beta_{t+1}(j) & for\, t = T-1, T-2..1,\ 1 \leq j \geq N \end{array} \tag{B.14}$$

Notice that the initialization $\beta_T(i) = 1\ for\ 1 \leq i \geq N$
is arbitrary defined, and β_T itself has not a proper meaning outside the context of the algorithm. Also in this case the complexity is $N \cdot T$

B.4 Key problems of interest

Given the HMM there are three fundamental problems to discuss:

- Compute the emission probability $P(S|M_\theta)$ of a given osservation S by a given model M_θ.
- Given the observation sequence S find an optimal sequence of underlying states
- How to adapt the HMM parameters to maximise the match with a given [set of] sequence.

The concepts of optimality and maximum match can vary between different applications. A solution for the first problem will be explained, being $P(S|M_\theta)$ used in the algorithms described in 5, where a solution to the third problem is showed. The second problem, although historically relevant is not considered here because it never occours during the development of the algorithms described in this work.

[5]computing α as sum of the probabilities over all possible path has complexity TN^T

B.4.1 The Forward-Backward procedure

The emission probability $P(S|M_\theta)$ could be explicitly expressed as

$$P(S|M_\theta) = \sum_{all\ X} P(S|X, M_\theta)P(X|M_\theta) = \sum_{x_1, x_2 \ldots, x_t} \pi_{x_1} b_{x_1}(S_1) a_{x_1 x_2} b_{x_2}(S_2) \ldots a_{x_{T-1} x_T} b_{x_T}(S_T) \tag{B.15}$$

This formula is unefficient if applied compute $P(S|M_\theta)$ because of the great computational complexity. A more efficent way to do this computation is to use the functions α and β defined in B.3. The formula to be applied is:

$$P(S|M_\theta) = \sum_{i=0}^{N} \alpha_T(i) \tag{B.16}$$

or, referring to β

$$P(S|M_\theta) = \sum_{i=0}^{N} \beta_1(i) \tag{B.17}$$

Appendix C

Principal Component Analysis

C.1 Definition

Principal component analysis is a method to simplify data. The basic aim of this procedure is the reduction of the number of variables (representing features of the analyzed phenomenon) representing them with a minor number of latent variables. It is possible through a variables linear transformation that projects them in a new reference system in which the new variable with the greatest variance is projected on the first axes, the one second for variance size on the second and so on. The complexity reduction consists in dropping a number new variables with minor variance, keeping just the *principal* ones.

Assuming that the original variables x have zero mean (it can be forced subtracting the actual mean) finding the principal component is a matter of finding the vector $\mathbf{w}_1$ from:

$$\mathbf{w}_1 = \arg \max_{\|\mathbf{w}\|=1} E\left\{\left(\mathbf{w}^T \mathbf{x}\right)^2\right\} \tag{C.1}$$

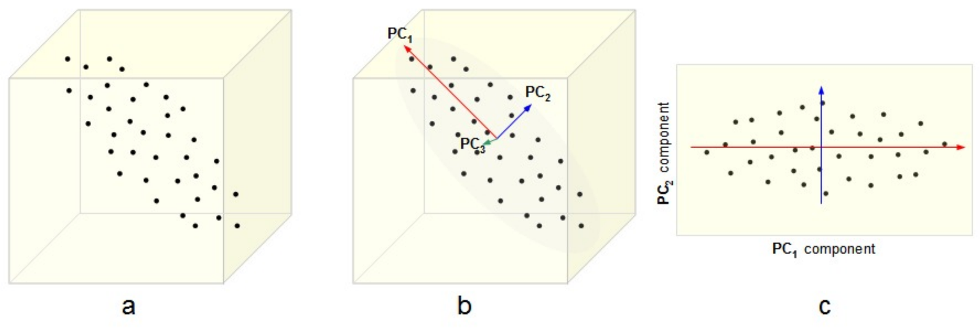


Figure C.1: Example of PCA

With the first $k-1$ components the k^{th} component can be computed subtracting the first $k-1$ principal component from X

$$\hat{\mathbf{x}}_{k-1} = \mathbf{x} - \sum_{i=1}^{k-1} \mathbf{w}_i \mathbf{w}_i^T \mathbf{x} \tag{C.2}$$

and substituting it:

$$\mathbf{w}_k = \arg \max_{\|\mathbf{w}\|=1} E\left\{ \left(\mathbf{w}^T \hat{\mathbf{x}}_{k-1} \right)^2 \right\}. \tag{C.3}$$

Original data are than projected in the obtained reduced vectorial space.

C.2 Using The Covariance Matrix

An easier way to compute the linear transformation is to use the covariance matrix of the data:

$$\mathbf{C}_x = E\left\{ (x-\mu_x)(x-\mu_x)^T \right\} \tag{C.4}$$

that is estimated as

$$\mathbf{C}_x = \frac{1}{n} \sum_{n}^{i=1} (x-\mu_x)(x-\mu_x)^T \tag{C.5}$$

where n is the number of observations considered. The eigenvectors of this matrix, arranged in rows, and ordered by the size of the associated eigenvalue are the transformation matrix needed.

Appendix D

Competitive Neural Networks

Competitive neural networks are adaptative classifiers. Their neurons distribute themselves to recognize frequently presented input vectors. The name ”competitive” address the fact that when an input vector is presented only the most activated neuron is considered ”the winner” and identifies the class to which the input is recognized to belong.

D.1 Architecture

Here is shown the network architecture as represented in Matlab. The ”Ndist”

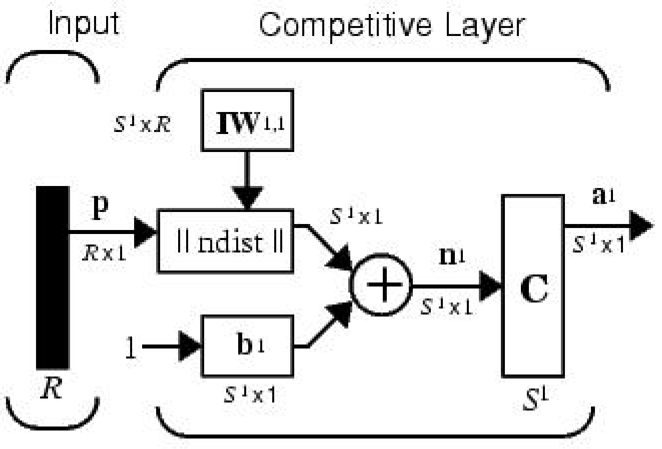


Figure D.1: Competitive Networks in Matlab

box in figure D.1 accepts the input vector p and the input weight matrix $IW_{1,1}$, and produces a vector having S1 elements. The elements are the negative of the distances between the input vector and vectors $IW_{1,1}$ formed from the rows of the input weight matrix. The net input $n1$ of a competitive layer is computed by finding the negative distance between input vector p and the weight vectors and adding the biases b. If all biases are zero, the maximum net input a neuron can have is 0. This occurs when the input vector p equals that neuron's weight vector. The competitive transfer function accepts a net input vector for a layer and returns neuron outputs of 0 for all neurons except for the winner, the neuron associated with the most positive element of net input $n1$. The winner's output is 1. If all biases are 0, then the neuron whose weight vector is closest to the input vector has the least negative net input and, therefore, wins the competition to output a 1[1].

D.2 Training

During the training sample inputs are presented to the net. For every sample presented the winner neuron is identified (on the basis of the weights at that phase of the training) and its weights are adapted using the *Kohonen learning rule*, already discussed in 2.4.1 that, with the notation used in the scheme in figure D.1, supposing that the i^{th} neuron wins, the elements of the i^{th} row of the input weight matrix is adapted with the following formula:

$$_iIW^{1,1}(t) =_i IW^{1,1}(t-1) + \alpha(p(t) -_i IW^{1,1}(t-1)) \tag{D.1}$$

where t stands for "time" and is the step of the training process.

Thus, the neuron whose weight vector was closest to the input vector is updated to be even closer. The result is that the winning neuron is more likely to win the competition the next time a similar vector is presented, and less likely to win when a very different input vector is presented.

D.3 Adjusting Biases

One of the limitations of competitive networks is that some neurons may not always get allocated. In other words, some neuron weight vectors may start out far from any input vectors and never win the competition, no matter how long the training is continued. The result is that their weights do not get to learn and they never win. These unfortunate neurons, referred to as dead neurons, never perform a useful function. To stop this from happening, biases are used to give neurons that only win the competition rarely (if ever) an advantage over neurons that win often. A positive bias, added to the negative distance, makes a distant

[1]extract from [Mat04]

neuron more likely to win. To do this job a running average of neuron outputs is kept. It is equivalent to the percentages of times each output is 1. This average is used to update the biases so that the biases of frequently active neurons will get smaller, and biases of infrequently active neurons will get larger. The result is that biases of neurons that haven't responded very frequently will increase versus biases of neurons that have responded frequently. As the biases of infrequently active neurons increase, the input space to which that neuron responds increases. As that input space increases, the infrequently active neuron responds and moves toward more input vectors. Eventually the neuron will respond to an equal number of vectors as other neurons. This has two good effects. First, if a neuron never wins a competition because its weights are far from any of the input vectors, its bias will eventually get large enough so that it will be able to win. When this happens, it will move toward some group of input vectors. Once the neuron's weights have moved into a group of input vectors and the neuron is winning consistently, its bias will decrease to 0. Thus, the problem of dead neurons is resolved. The second advantage of biases is that they force each neuron to classify roughly the same percentage of input vectors. Thus, if a region of the input space is associated with a larger number of input vectors than another region, the more densely filled region will attract more neurons and be classified into smaller subsections.

Bibliography

[AC00] Horst-Micheal Gross Andrea Corradini. Imlementation and comparison of three architectures for gesture recognition. *IEEE*, pages 2361–2364, 2000.

[AJ05] Sébastien Marcel Agnès Just. Two-handed gesture recognition. Technical report, IDIAP, may 2005.

[AS] Antonina Starita Alessandro Sperduti. Supervised neural networks for the classification of structures.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[Bri90] John S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. pages 211–217, 1990.

[Cho90] Yen-Lu Chow. Maximum mutual information estimation of hmm parameters for continuos speech recognition using the n-best algorithm. *IEEE*, pages 701–704, 1990.

[G.E] B. J. McCarrangher G.E.Howland, P. Sikka. Skill acquisition from human demostration using a hidden markov model.

[JP] Mikko Kurimo Janne Pylkkonen. Duration modeling techniques for continuous speech recognition.

[JS02] Massino Bergamasco. Jorge Solis, Carlo A. Avizzano. Teaching to write japanese caracters using a haptic interface. *Proceedings of the 10th Symp. On Haptic Interfaces For Virtual Envir. & Teleoperator Systs.*, pages –, 2002.

[Koh90] T. Kohonen. The self-organizing map. *Proce. IEEE*, 78:1464–1480, 1990.

[LDD] Angelo M. Sabatini Laura DiPietro and Paolo Dario. A survey of glove-based systems and their applications.

[LR86] B.H. Juang L.R. Rabiner. An introduction to hidden markov models. *IEEE ASSP Magazine*, pages –, 1986.

[Mat04] Mathworks. *Matlab manual*, 2004.

[MB97] Alex Portland Mattew Brand, Nuria Oliver. Coupled hidden markov models for complex action recognition. *IEEE*, pages 994–999, 1997.

[Mic05] Alessio Micheli. Hmm, just an introduction of the model, 2005.

[MMC05] Judy M. Vance Melinda M. Cerney. Gesture recognition in virtual environments: A review and framework for future development. Technical report, Iowa State University, March 2005.

[OPR] E. Sotgiu S. Pabon A. Frisoli J. Ortiz M. Bergamasco O. Portillo-Rodriguez, C.A. Avizzano. A wireless bluetooth dataglove based on a novel goniometric sensors.

[PF97] Alessandro Sperduti. Paolo Frasconi, Marco Gori. A general framework for adaptive processing of data structures. pages –, 1997.

[Ros96] I. Rossitto. Identificazione del gesto tramite hidden markov model. Master's thesis, University of Pisa, Faculty of Computer Science, 1996. (Italian Thesis: Gesture Recognition through Hidden Markov Models).

[Vap99] Vladimir N. Vapnik. An overview of statistical learning theory. *IEEE Transaction on Neural Network*, 10(5):988–999, September 1999.

[WB] Yves Guiard Abigail Sellen Shumin Zhai William Buxton, Mark Billinghurst. Human input to computer systems: Theories, techniques and technology. Available on http://www.billbuxton.com.

[WZ02] Xiang-Long Tang Wei Zhao, Jia-Feng Liu. On-line handwritten english word recognition based on cascade connection of character hmm. *Procedings of the first International Conference on machine Learning and Cybernetics, Beijing*, 1:1758–1761, November 2002.

[YA97] H. Yin and N.M. Allison. Bayesian learning self-organising maps. *Electronics Letters*, 33:304–305, 1997.