〔公　開〕

TR－C－0159

An indirect approach to hand gesture recognition for applications combining hand gestures and natural language

デュック　トゥリヌ　　ジュリ　ティヘリノ
Duc TRINH　　Yuri TIJERINO

１９９６　３．１５

ＡＴＲ通信システム研究所

An indirect approach to hand gesture recognition for applications combining hand gestures and natural language

Duc C. TRINH**, Dr Yuri A. TIJERINO*

* ATR Communication Systems Research Laboratories
2-2 Hikaridai, Seikacho, Sorakugun, Kyoto 619-02 Japan

** Ecole Nationale Superieure des Telecommunications
46, rue Barrault, 75013 Paris, France

** E-mail: trinh@email.enst.fr
* E-mail: yuri@atr-sw.atr.co.jp

Contents

Abstract

This report introduces an indirect approach to hand gesture recognition for applications combining hand gestures and natural language. In order to build an intuitive 3D GUI, we plan to use natural language. We believe that hand gestures are an important non verbal method for augmenting verbal communication, we intend to integrate a hand gesture interpretation module with a 3DGUI to resolve the inherent difficulties and ambiguities of processing natural language. This report describes a study on natural hand gestures useful for manipulating, modelling and translating 3-D virtual objects. From experimental analysis we divided these gestures in two categories. One category refers to gestures used to interact with virtual objects directly in a single mode (e.g., grabbing), while the other category usually combines information from other modes, such as natural language and can be useful for multimodal interface. Usually several getures associate with a single operative concept. Direct, template-based, approaches provide robust gesture recognition for a small set of gestures, but are not intuitive to use because gestures usually differ from person to person even for a single concept. Therefore we employed an indirect approach based a neural networks implementation that determines the position of a focal point generalized from experimental results. The focal point can be defined as the point on the hand where the attention of a person viewing the gesture is focused. We found that observation of this point gives sufficient information for interpreting some gesture's intententions.

I. Introduction

Interfaces between human and machine evolved from 2D Graphic user interfaces (GUI) and are about to enter the Multimedia age. Icons, windows, menus and so on, are 2D GUI elements which are now commonly used with most computers. However, 2D GUI such as X-Windows, Microsoft Windows and Macintosh GUI require the use of a pointing device, such as a mouse, to provide inputs to the computer.

Recently, the evolution of hardware technology gives us the opportunity to add one more visual dimension to human-machine interactions. With the help of a 3D graphical environment, these interactions can get closer to human-human communications. An application such as the Virtual Space Teleconferencing system, [5] developed in ATR-CSRL, uses 3D GUI in order to build a virtual workspace shared by several users in different locations and provide realistic sensations. However, the

technology hasn't reached its maturity yet, therefore we still expect a lot of interesting research on the subject.

GUI's can obviously benefit from hand gesture interpretation as a next-generation kind of input devices to handle 3D information. Originally, S. Sidney Felds built a Glove-Talk system using neural networks and a Dataglove [4] to translate sign language gestures. Most recently, non-intrusive techniques are also being developed to recognize hand gestures using video cameras [3], and combined with Hidden Markov Models [17][10]. Baudel and Beaudel-Lafon integrated a hand gesture interpretation system for computer-aided presentations. The system could recognize 16 gestures and allowed the user to navigate within a Hypercard based presentation. With the Virtual Space Teleconferencing System, users can manipulate objects thanks to a simple grab-release gesture recognition module.

On the other hand, speech recognition has been a major theme of research. Though there is still much to expect in robustness and flexibility, the Virtual Space Teleconferancing System developed at ATR already includes a speech recognition module. Efforts are made to use speech recognition to interactively generate, manipulate and Modify 3-D Shapes (The What-you-say-is-what-you-see (WYSIWYS) framework [12]). The system is expected to employ deformable superquadrics [8] to build a framework for interactive indexing of knowledge-level descriptions [11] of human intentions to a symbol-level representation. However, due to the ambiguity of operation concepts, users could combine hand gestures naturally with speech to clarify verbal descriptions. Therefore we tried to understand the relationship between hand gestures and natural language for our system. The system is aimed to be robust enough to be used by any kind of users, especially those who don't know anything about virtual reality, CAD applications and neural networks.

II. Hand gestures and natural language in 3D GUI

Usually, speech and hand gestures are treated independently. A few researchers tried to combine speech and hand gestures for a more convivial interface. For instance, David Weimer and S. K. Ganapathy combined speech and gestures in a system to create a synthetic visual environment [15] where one could modify 3D mathematical shapes. This CAD application implemented the same capabilities as the wand of Vickers and Clark [13] [2] but used the index finger tip like a stylus for locating point of the interest. Thumb abduction or voice commands initiated the pick. The gesturing part was reduced to a small set of hand gestures while speech was used

to invoke system commands. Since they could be easily recognized, special thumb gestures were used.

Our approach is different. Most of the interfaces using hand gestures need a special set of hand gestures. Therefore, one has to learn these particular predefined gestures before using the system. We believe it is possible to go one step further toward a more intuitive interaction with the machine. By studying hand gestures, we should be able to understand them better. We will then be able to build a system that understands the intention of the gesture instead of just matching hand shapes to predefined templates.

We plan to integrate this hand gesture interpretation module with ATR's Virtual Space Teleconferencing System. People could then model objects with 3D shape ontologies and implicit functions such as superquadrics to obtain inmediate visual feedback. The system should be able to translate intentions, given by verbal instructions, of the user accurately into a graphical transformation on objects. Because of the ambiguity of natural language, we want to use a hand gesture recognition system to complement verbal commands and help the user communicate with the computer.

III. Understanding hand gestures

A. An experiment to clarify the relationship between hand gestures and natural language

We used Thingworld [9], an experimental software that implements superquadrics, to see which commands were the most useful to model objects. These commands are the most basic commands for a CAD application using implicit functions. We tried to find out what kind of gestures was naturally associated with these commands.

The experiment was performed in three parts. In each part, we presented several scenes to the subject. The first part dealt with transformations: eleven 3D graphical scenes, each scene corresponded to one specific transformation, were shown using WorldToolKit [16], a library of C functions with which one can build virtual environments where graphical objects simulate real world properties and behavior:

- Scene 1 was Vertical stretching
- Scene 2 " Horizontal stretching

- Scene 3 " Local elongation
- Scene 4 " Horizontal squeezing
- Scene 5 " Local shrinking
- Scene 6 " Vertical squeezing
- Scene 7 " Twisting
- Scene 8 " Bending
- Scene 9 " Pinching
- Scene 10 " Extruding
- Scene 11 " Flattening

The second part dealt with motion: translation and rotation. We tried to find out what gesture would be done to move objects in a 3D environment.

The third part dealt with the deictic gestures. Using both speech and hand gestures, the user was asked to select a particular object among several presented with WorldToolkit.

We had 11 subjects for our experiment, 5 were female and 6 were male. They came from various countries such as Marocco, France, Japan, Canada and The United States. The aim of the experiment is not to do a survey of all the possible hand gestures associated with the formely mentioned transformations but it gives us interesting sample data. The data we collected was interesting enough for our purpose and there are a full range of possible analysis that can be done.

B. Analysis

Robustness has to be one of the main attributes of our recognition system. This implies limitation. Therefore we had to focus on some particular transformations. However, even with these restrictions, the experiment showed us that a system that recognizes natural gestures should deal with many different hand gestures.

After analysis of the data (the experiment was stored on video tapes), we found that it was possible to distinguish two categories in the hand gestures done by the subjects. The first category was those gestures in which the subject behaved as if the object really existed and operations could be performed on it as if it was made of clay. The second category consisted of those gestures in which the subject behaved as if the object could be directed to perform some operation being guided by hand motion.

We coined the term clay paradigm to explain the behavior in the first category.

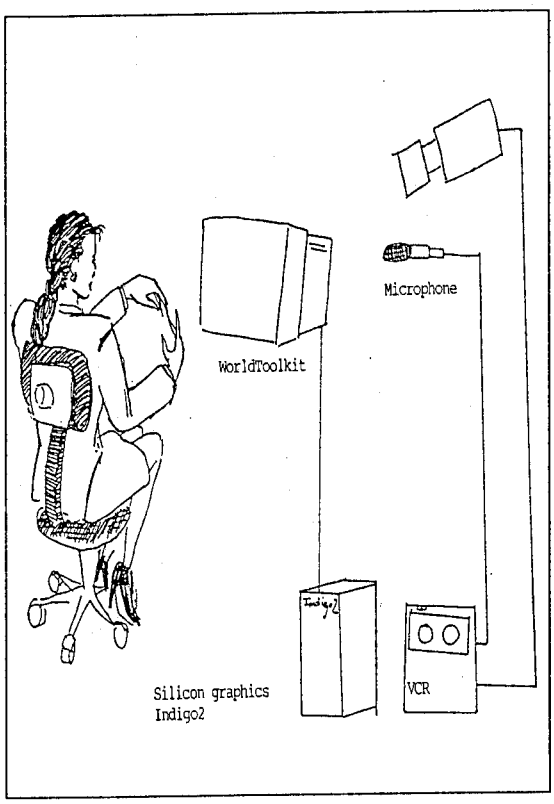


Figure 1: Experiment to clarify the relationship between hand gestures and verbal command

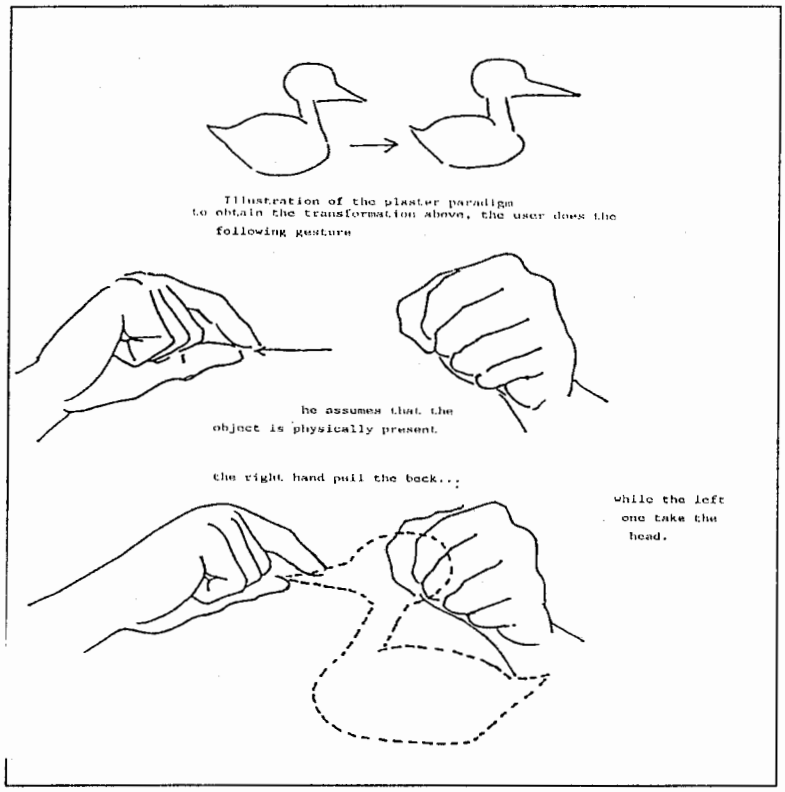


Figure 2: Illustration of the clay paradigm gestures.

Given this definition, we classified the gestures in two categories; the gestures using the clay paradigm, and those which correspond more to a command.

Using the clay paradigm gestures, the subject considered that he could actually touch the virtual object in order to deform and modify it. Since it was displayed in 2D on a screen, the subjects had to mime a physical interaction with the object (figure 2).

The second type of gestures, command gestures, corresponded more to a command given to the computer, we called them the command gestures. For these, the shape of the object did not really influence the gesture. Actually, these hand gestures would have been the same for a simple geometric figure such as a cube or a cylinder. The virtual state of the object is considered when doing the gesture. Figure 3 illustrates the difference between the two categories.

The difference between these gestures is like the difference between making your own cooking and order something at a restaurant, the result (a full stomach) is in both case the same.

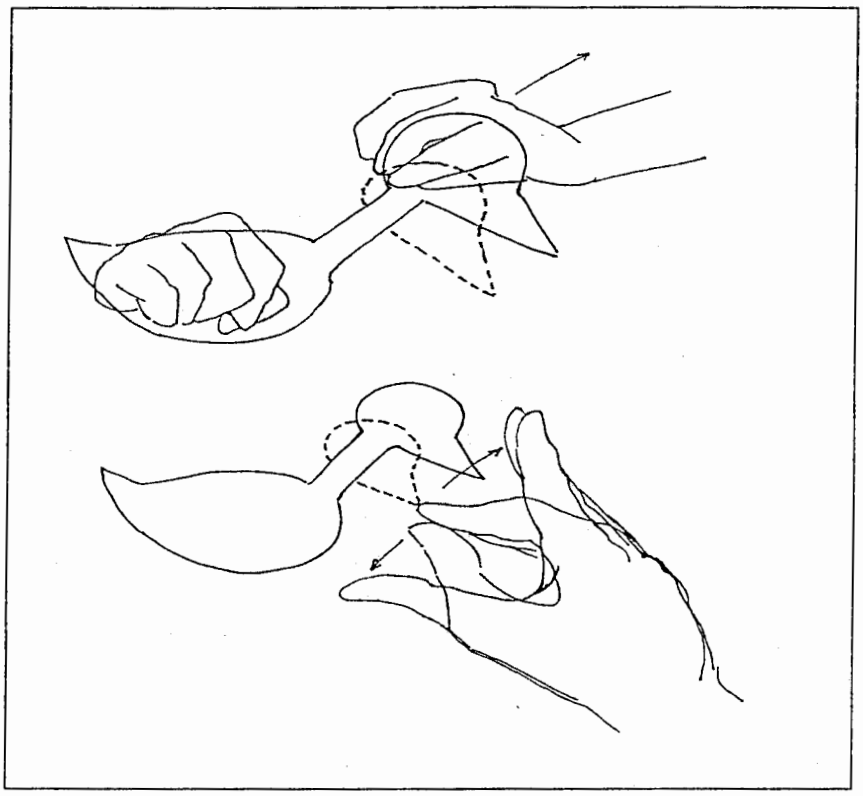


Figure 3: Difference between Clay paradigm gestures and Command gestures.

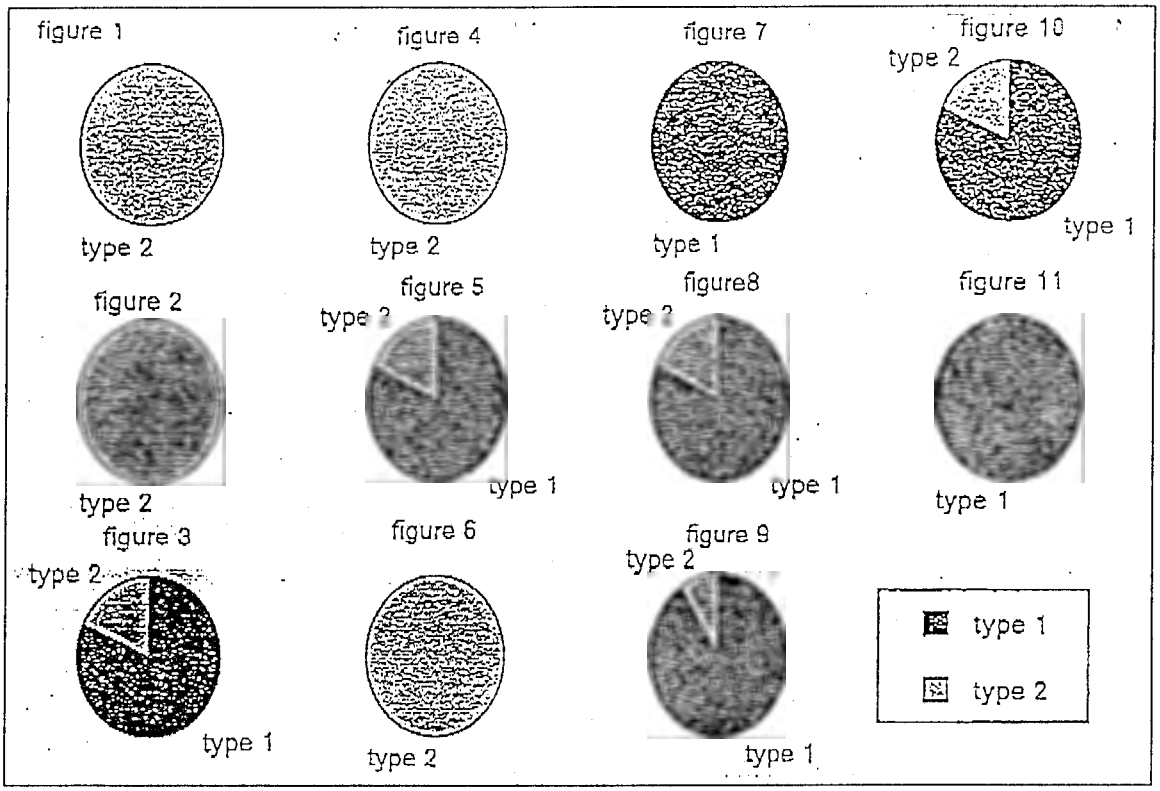


Figure 4: Classification of the gestures for each scenes.

For each transformation we collected the data and then separated them classified them in these two types (See figure 4). It appears that for some scenes, particularly those concerning a transformation of the whole object, only command gestures were done. Though figure 5 shows that clay paradigm gestures are preferred, subjects tend to modify a virtual object as if it was maleable, most of the time, when clay paradigm gestures were used, command gestures could have also been used (see figure 4).

As we saw, clay paradigm gestures suppose physical contact with the object. To accurately transform the object according to the hand gesture, the user would need to have some kind of feedback, whether visual or even better, physical. The response on the object should also be done according to the clay paradigm. Besides, the system should simulate physical properties, have a collision detection system, have precise devices, and also have a accurate graphical model of the hand. Since this is not between the aim of this research, we focused our work on command gestures. An interesting point about command gestures is that, with these gestures, the important

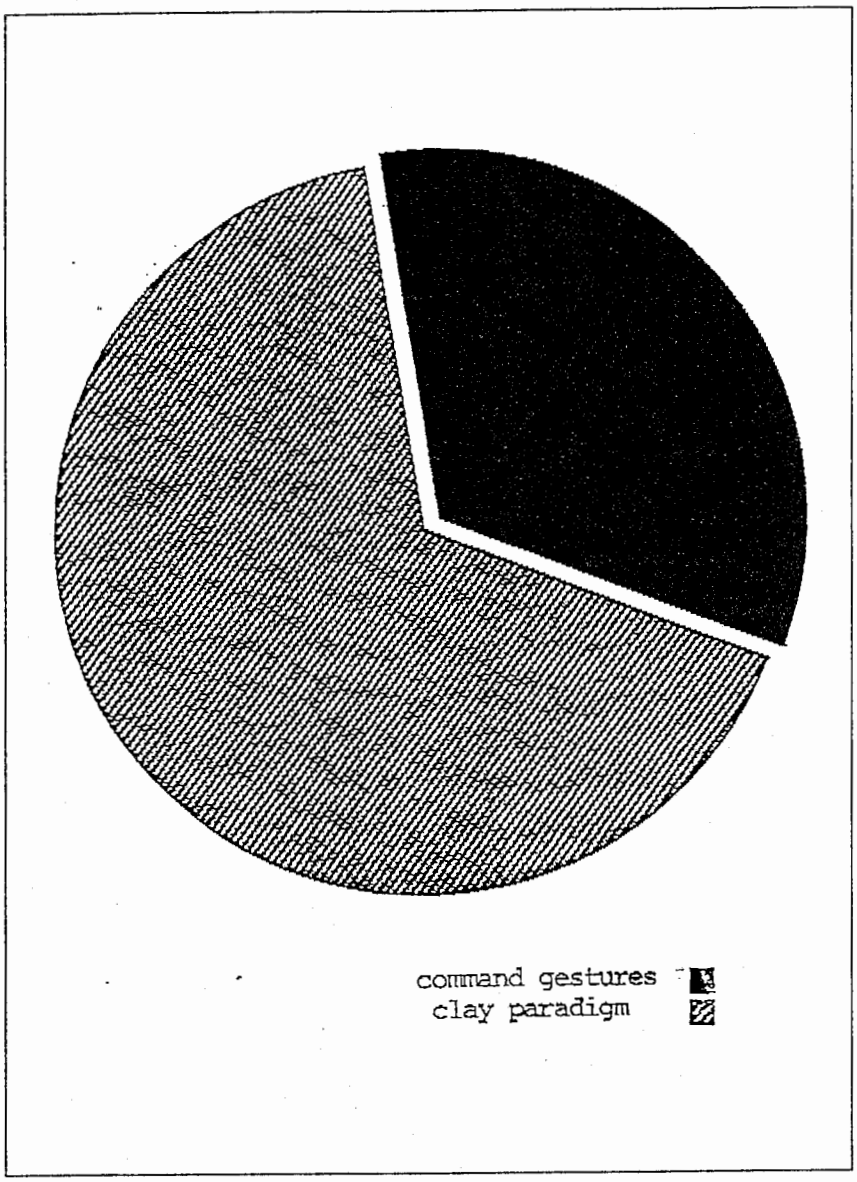


Figure 5: Clay paradigm gestures vs. command gestures.

thing is to recognize the meaning of the gesture more than its shape. So, with a system that can recognize the intention which lies behind the gesture, we should be able to deal with natural command gestures. Any user would be able to use the system in a very intuitive way without having to learn a special set of hand gestures.

IV. An indirect approach for hand gesture recognition

A. The focal point

The hand gesture recognition systems developed until now are bounded by the shape of the hand. That is, they are template-based and work at a low level of description. What we found out during our experiment is that, even for a simple task as rotation, subjects used very different gestures in the sense that the movement of the arm and the shape of the hand could be very different. Still, for all of these, an external observer could understand what transformation the hand gesture corresponded to. The gestures actually had the same meaning, they described the same concept. Therefore, template-based hand gesture recognition is not appropriate for recognizing natural gestures even if an infinite number of template is available to recognize all possible gestures. This is why we needed to recognize the command gestures at a higher level of abstraction. In this paper we propose to adopt an indirect approach for hand gesture recognition in order to sidestep the template-based approach and go directly to the concept described. This approach will be indirect because we are less interested in how the fingers are bent or how the arm moved in the space than in knowing the information brought by the gesture. Therefore the system won't tell us if the hand has a fist shape or if one of the fingers is raised but instead will tell the computer how the gesture is used to give its information.

As we observed the command gestures, we came to the conclusion that we could abstract the information contained in each gesture to the observation of a single point. For instance, though the gestures used to rotate were quite different, there was a common point that could be abstracted from all of these; some part of the hand was rotating or describing a revolution (see figure 6 and figure 7). From this part we could abstract the most meaningful point whose trajectory and changes of orientation give information for the whole gesture; This point contains all the necessary information of the hand shape: We called it the focal point.

Intuitively, our attention, during the gesture is focused on a certain part of the hand, depending statically on the shape of the hand. That is where we located

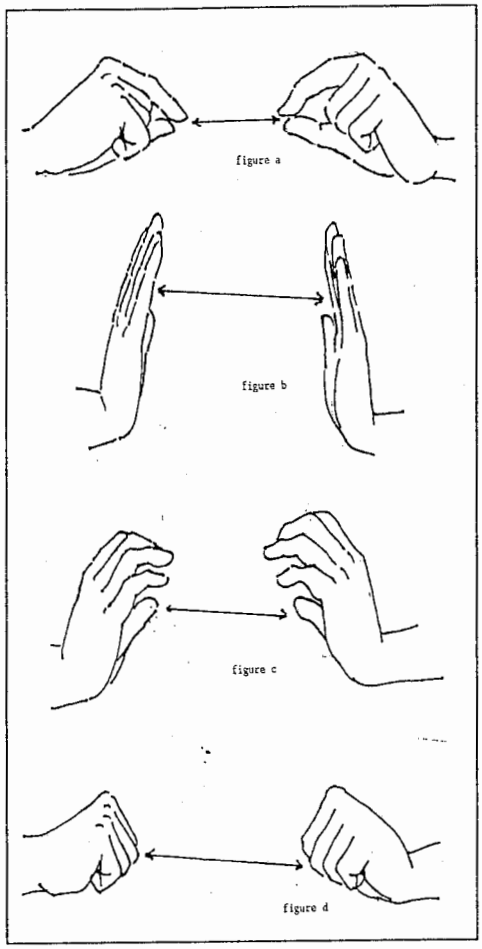


Figure 6: Different gestures for the same concept.

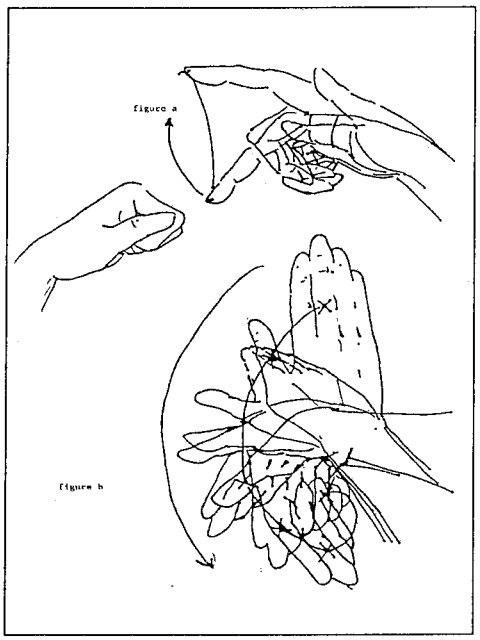


Figure 7: Different gestures for a same concept.

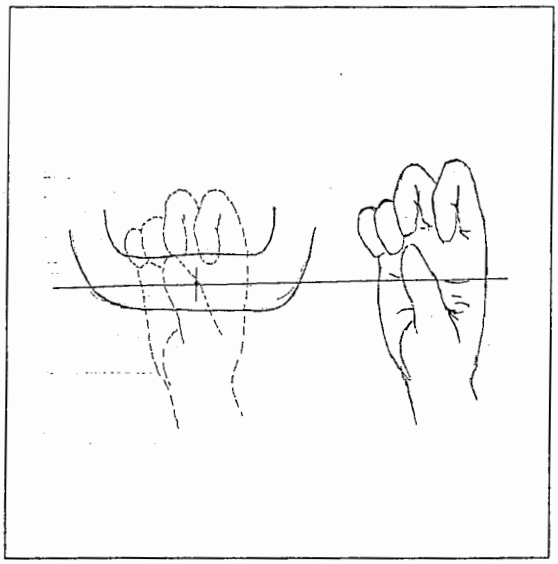


Figure 8: The attention is focused at the center of gravity of the handler.

the focal point.

The shape of the hand might be used to indicate a position as it is the case for deictic gestures, where the attention is focused on a finger's tip, or the hand might be used to represent the object, or the subject might imagine handlers on the object to grab, in this case, the attention is focused on the center of gravity of the imaginary handler (see figure 8) and the hand gesture means that you are ready to move something. So the attention is not systematically focused on the index finger tip nor on the palm, it varies according to the hand shape.

These hand shapes depend on how we usually use our hands. As an example, when the thumb tip and the index finger tip touch each other, the hand gesture that is done when you are holding something very thin between your fingers, like a rope, the attention is focused on the supposed contact point, where the object is supposed to be (see figure 9), this hand gesture means that you want to express the concept "pull" what else can you do with a rope? push it? . Actually, to clarify this point we should do some more experiments (see recommendations).

In some cases, though, if only one hand is used, the thumb and the other part of the hand can play different role as it is the case for one of the enlargement gestures. In this case two parts of a hand would play the same role as two hands, thus, with the same method, two focal points would have been necessary. We could integrate the enlargement information in the system. Mourouvapin et al. have worked on this

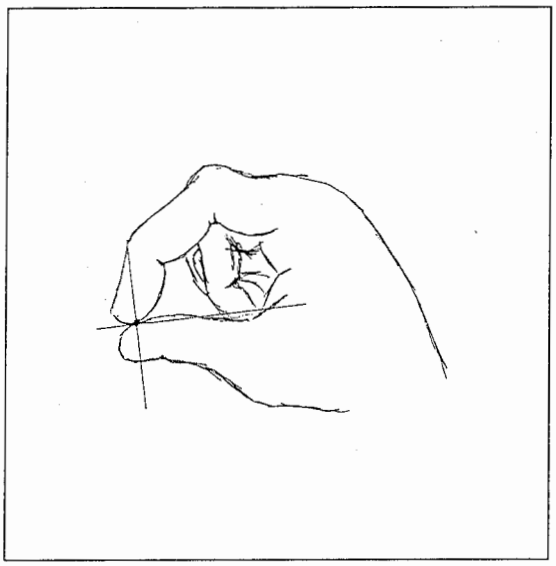


Figure 9: An illustration of the focal point.

particular gesture using neural networks [7], so we shall not work on these gestures.

We insist on the fact that finding the focal point is only one of the possible indirect approaches for hand gesture recognition, this method is valid for several concepts but not for all. In this paper, we just want to demonstrate that such a method is tractable.

B. Using neural networks

Neural networks have already been commonly used for recognition. S. Sidney Felds used several neural networks for his Glove Talk system, one to detect the beginning of a gesture, it had as output 1 or 0, one to recognize the hand shape for root word, it could recognize 66 different hand shapes, and networks for *ending, rate and stress.* The neural networks were used as Classifiers, so they were trained to determine in which class a gesture could be (if it means one of the 66 possible words for instance). Mourouvapin et al. used recurrent neural networks because of their ability to handle time, but they used them in order to calculate the enlargement corresponding to the gesture. Actually neural networks can be used for calculation and this is how we want to use them.

Since we want to calculate the position of a particular point of the hand, it is possible to calculate only the relative position to the position of the wrist. Due to physical reasons (the wrist joint limit), this position is bounded by what we called the bounding box (see figure 10 11) which surrounds the wrist and has faces that are perpendicular to the forearm, such a system is not chaotic so the network has a chance to converge.

Then if we train the network properly, with different hand shape i and the corresponding position p_i, neural network has the ability to find a function f so that:

for all i, $f(handshape[i]) = p_i$;

The problem is then to find the correct function f since we are also interested in the intermediate values of f. There is no explicit function that could give the position of the focal point. It is not always located on the finger tip for instance, it may even not be on the hand at all. It depends on knowledge such as when raised alone, the index finger indicates a position, or the fact that we use different hand shape to grab or to push, where we might use all our fingers. We need to use a system that could learn this knowledge. Neural networks provide this learning ability. Thus we propose to use them in our system, to calculate the position and orientation of the focal point.

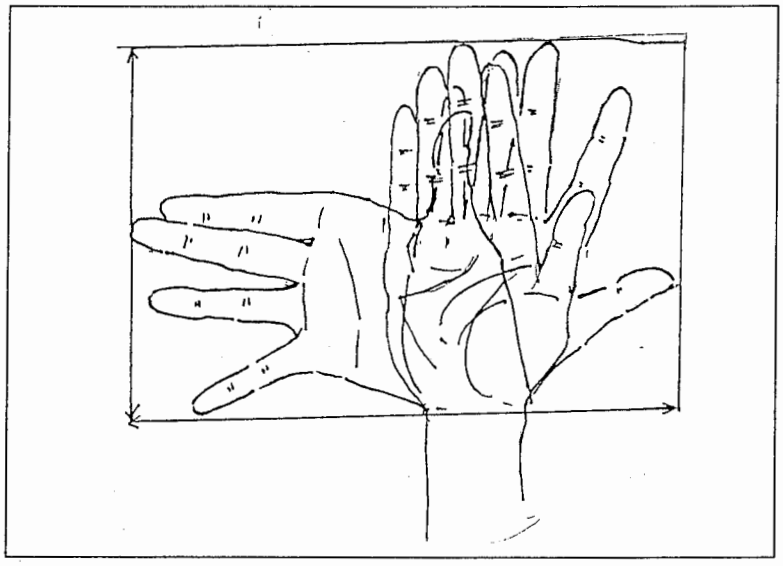


Figure 10: front view of a joint limit.

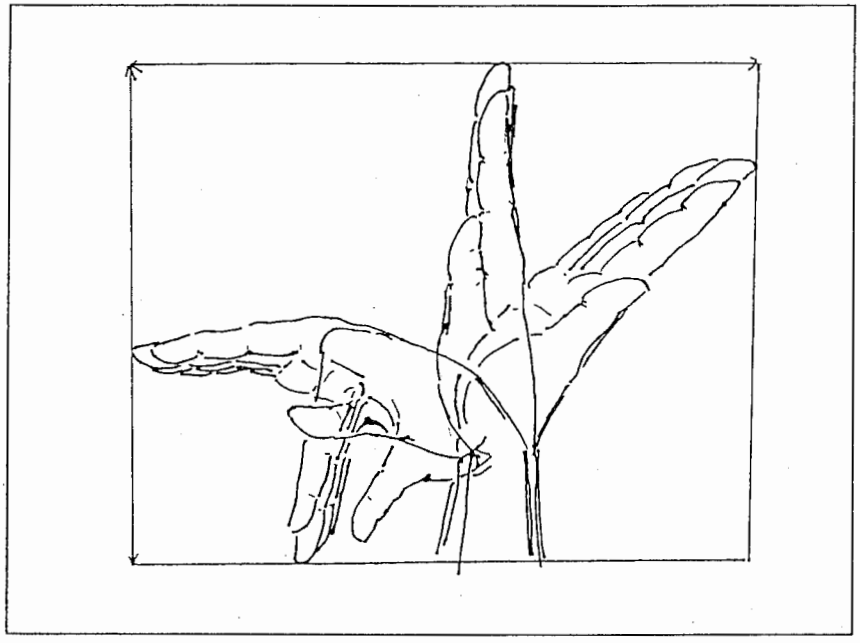


Figure 11: Side view of a joint limit.

The main difference between this approach and the template-based approach is that we don't need to define a value for all the position of the fingers or for all the positions of the wrist that we want to recognize. We give only some particular points of a function, enough to obtain a good approximation. This is the reason why we needed to teach the networks some basic knowledge, but we didn't want to recognize only the points we defined but the global function.

V. Building the system

A. The Cyberglove

In order to give hand shape information to the computer, we used a CyberGlove developed by Virtual Technologies [14], this instrument has 18 sensors added to a Polhemus Fastrak to measure with relative accuracy the movements of the fingers and the hand.

The Fastrak and the Cyberglove are connected separately to the computer (see figure 12) so you can handle information coming from these different sources independently. When the user is wearing the glove, the Polhemus Fastrak is located on the forearm and so its position doesn't change with the hand shape. In this paper, we will consider that this position corresponds to the position of the sensor. Figure 13 shows the position of the 18 sensors on the palm and fingers.

We used the CyberGlove with SunMicrosytems RPC (Remote Procedure Calling)[1]. RPC enables us to use the data of a CyberGlove connected to another machine. On the machine where the CyberGlove is connected, a server could be run in a continuous loop. Then if we need the data, we just have to run a client program that asks for data to the server (see the programs at the end of the paper).

This program takes the information given by the glove and then convert them into angle information according to the calibration file. The data format sent by the server is the following:

```
angles (in order):
finger 0 :
thumb MPJ (joint where the thumb meets the palm)
thumb IJ (outer thumb joint)
```

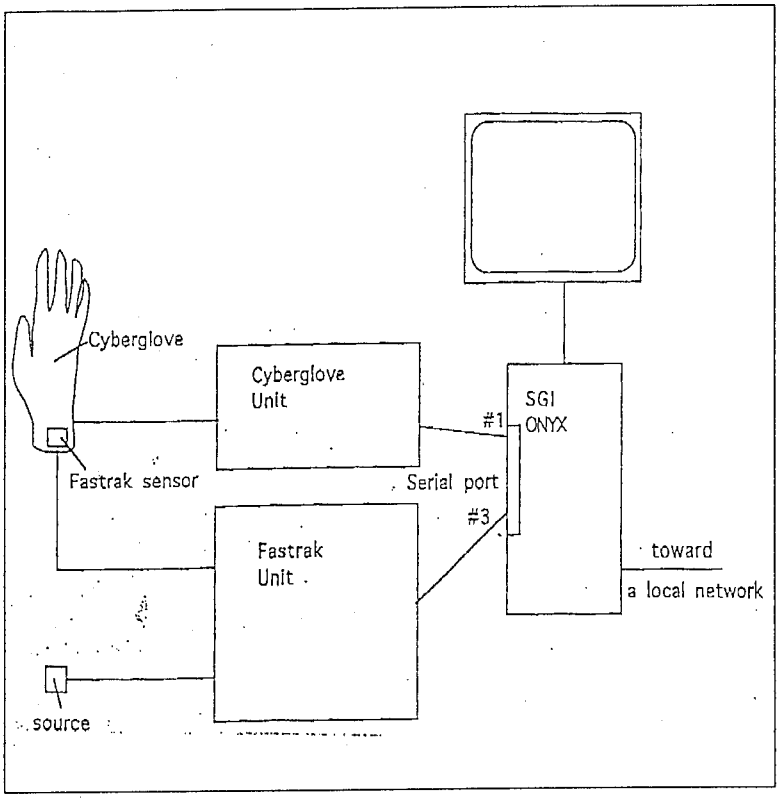


Figure 12: Device used for the system

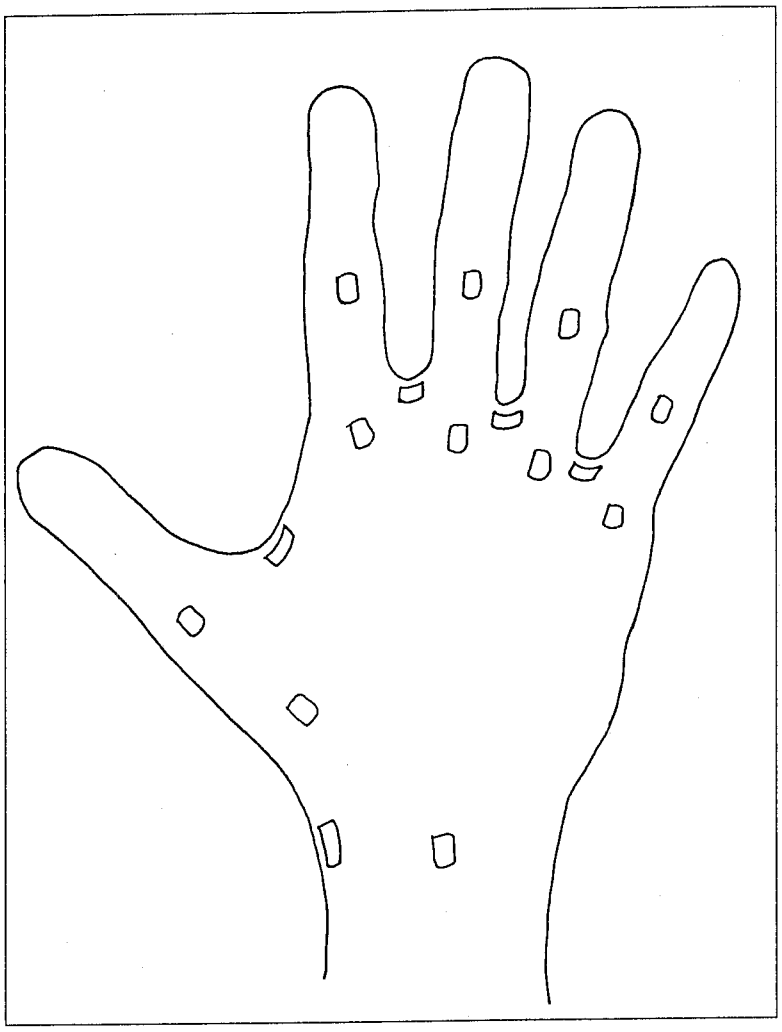


Figure 13: Position of the 18 sensors of the Cyberglove

```
finger 1 :
index MPJ (joint where the index meets the palm)
index PIJ (joint second from the finger tip)

finger 2 :
middle MPJ
middle PIJ

finger 3 :
ring MPJ
ring PIJ

finger 4:
pinkie MPJ
pinkie PIJ

space:
thumb abduction (angle between thumb and index finger)
middle-index abd'n (angle between ring and middle fingers)
ring-middle abduction (angle between ring and middle fingers)
pinkie-ring abduction (angle between pinkie and ring finger)

wrist:
wrist pitch
wrist yaw
```

You may notice that there are only 16 angles for 18 sensors. It is because the server we used doesn't handle neither the the thumb rotation (angle of thumb rotating across palm) nor the palm arch (causes pinkie to rotate across palm). For more accuracy, it may be useful to include this information in our system and so to rewrite the server program for the Cyberglove.

To get the absolute position of the wrist, we use the Pohlemus Fastrak attached to the CyberGlove. The system has two components:

- A source, which emits electromagnetic fields. In the configuration we used, the source was attached to a wooden table in front of the user.

- A sensor, which responds to the source's electromagntic fields according to its position to the source. The sensor feeds the Control Unit with six analog signals, which describe the sensor's coordinates (x,y,z) and orientation (azimuth (Ψ), elevation

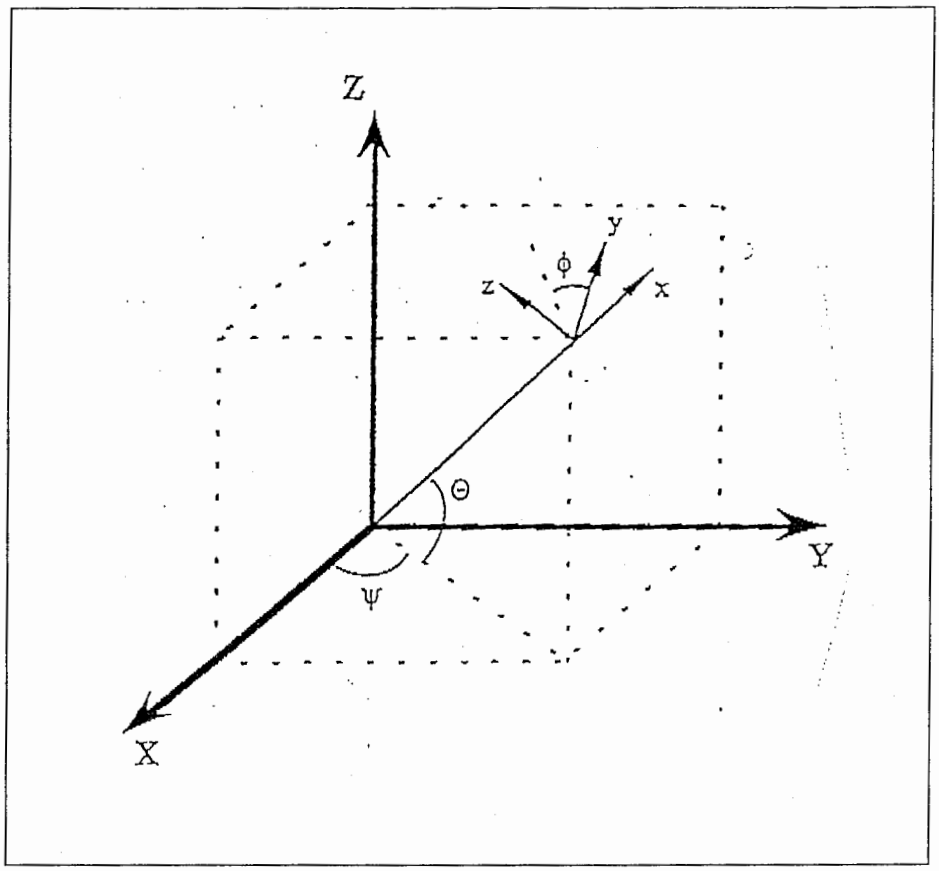


Figure 14: Euler angles for the Fastrak

(Θ) and roll (Φ): Euler angles). (see figure 14)

In our system, we didn't use RPC for the Fastrak but instead, since World-ToolKit could handle this sensor, we preferred to use the built-in programs (see the code at the end of the paper).

B. neural networks

1. Aspirin/MIGRAINES

To build neural networks, we used Aspirin/MIGRAINES developed by the MITRE Corporation [6]. This software is aimed to help building many kinds of neural networks. It provides two parts:

- The Aspirin Language (files named xxx.aspirin) is a declarative language used to specify neural networks architecture. With this software, the user can describe his neural network at a very high level, compilers provided with the language handles the lower level part by generating a code, which can be linked into a more application-specific system.

- You can use the MIGRAINES interface to simulate the behavior of the network or to generate files that are readable by such a software as GnuPlot or Math Lab. You can also use directly the executable file generated by the compilers. This way you can train the network, check its output, compare the ouput to the target file, modify the learning rate and the error boundaries (see codes at the end of the paper).

2. The training data

We needed data to train the Network so we defined 92 different gestures (see in Annex), that constitute a set of points on the function we want to recognize; they are mostly hand gestures that correspond to the different joint limits but we defined these gestures by trying to guess what would be the behavior of the Network given this data.

For instance, to lower the influence of the thumb in the recognition process, since the information given by raising the thumb could be contained in the orientation of the focal point, we reproduced gestures twice, with the thumb raised and bent, both having the same output. For the training, this should be taken into account and thus, should lower the influence of the thumb abduction for the output.

Also we tried to define the gestures according to the joint limit, hoping that the network will interpolate correctly between two end position of a joint. Since we had to define manually the position of the focal point for each training gesture, for instance, for gesture n.1 we calculated that the position of the focal point is x=3, y=18 and z=0; we added the target output to the recorded data coming from the sensor. Now the calculations of these different positions have been done, to build more training data, it is easy to write a small program that could record the data of hand gestures and combine it with the data corresponding to the predefined positions of the focal point. For instance, we could ask several persons to do several times gesture n.1, then the program could associate each with 3 18 0.

Figure 15 shows the axis and the angles we used to calculate the relative position of the focal point to the wrist. Figure 16 gives an example of the data we arranged.

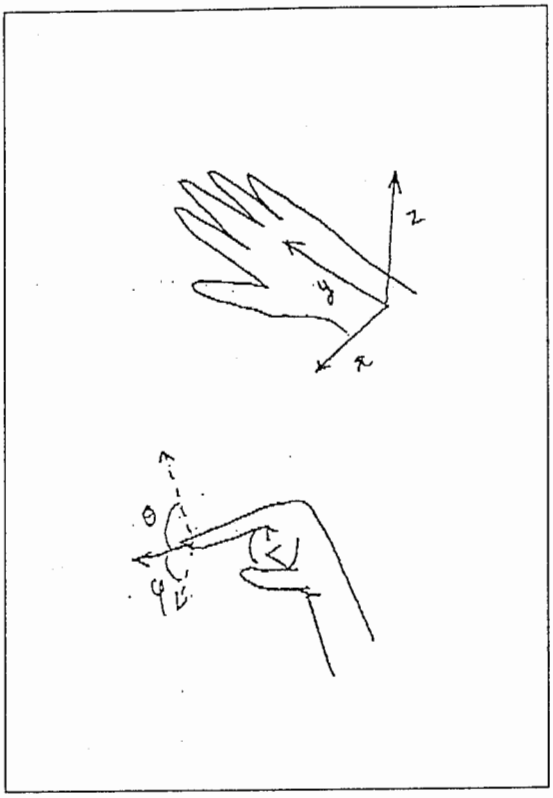


Figure 15: Axis and Angles used for the position of the focal point.

```
/* Case 1 */
82 107 95 113 88 114 96 62 65 73 13 -16 17 1 0 0 -67 0
3 18 0 0 0 0

/* Case 2 */
103 109 107 107 86 109 100 107 79 74 22 -18 12 8 0 0 -70 9
1 9 3 0 0 0

/* Case 3 */
96 102 101 124 86 122 93 109 72 81 18 -27 1 -2 0 0 -66 12
1 15 0 0 0 0

/* Case 4 */
107 93 103 114 86 108 100 98 81 74 17 -20 10 0 0 0 -67 6
1 11 5 90 0 0

/* Case 5 */
105 106 103 120 84 109 99 88 88 74 16 -22 8 -1 0 0 -68 8
3 0 18 90 0 0

/* Case 6 */
107 96 108 116 86 116 100 113 85 74 18 -22 9 0 0 0 -68 5
1 -1 9 90 0 0

/* Case 7 */
107 113 102 116 82 106 98 78 85 74 18 -17 10 6 0 0 -67 13
1 0 15 90 0 0

/* Case 8 */
103 107 102 107 85 114 100 108 87 67 17 -17 9 8  0   0 -67  10
1 -4 11 180 0 0

/* Case 9 */
 99 105 101 109  82 106  99 106  80  75  25 -20 8 2 0 0 -67  10
3 12 11 -45 0 0
```

Figure 16: Sample data with the angles given by the Cyberglove server and the relative position to the wrist.

neural network	NN for X	NN for Y	NN for Z	NN for angles
input nodes	16	16	16	16
hidden layer nodes	21	25	23	33
output nodes	1	1	1	2

Figure 17: Architecture of the neural networks.

However, the data have to fit the required format of the nodes' input. We distributed the values between -1 and 1. The values should not be too close from each other. We decided to normalize the values but there are other ways to handle it. Some gestures may have been reproduced several times according to the weight we wanted to give them in the recognition process.

For our system, we used simple 3 layer neural networks (*Perceptrons*) and use the backpropagation algorithm. We used one single node for each coordinate because of the continuity of the output. Due to the low number of training data, and for easier convergence and training we decided to divide the problem. So we built four different networks, one for each coordinate, and one for each angle. They are smaller than a single big network with 3 outputs so the number of weights is not too large compared to the number of training data. Figure 17 shows what architecture we finally decided to have.

3. Some convenient representations to deal with neural networks

The training stage is mostly hacking and you just have to learn from experience. No theory will tell you how many nodes the hidden layer should have or what learning rate will give you the best result. To visualize how we should modify the different parameters, this is how we considered neural networks (this is just an image to help people understand how the neural network behave, even if actually the theory for the perceptron is quite simple).

We represented a neural network as an array in the space. Its coordinates changed as its different weights changed. The amount of the change is related to the learning rate. We supposed that there were different points of convergence in the space, depending on the architecture and on the training data, each with a particular area of attraction. If the Network couldn't converge, there was none, if it was hard to make it converge, the areas of attraction were very small. The neural network was attracted by all of them. Once it got close enough to one of these point, we decreased the learning rate so that this point should have come quickly to its stable position.

The problem is that this point may not be the right one. In that case, whether you increase the learning rate in order to make the neural network "jump out" of an area of attraction or you start again your training from an earlier configuration of the Network, changing the learning rate. If none of these convergence points are good enough, you should change the number of nodes in the hidden layer.

To see if the function realized by the network is correct, you can either record other hand gestures as test gestures and then compare the outputs with the expected results, MIGRAINES can then show you the error of the network, or you can test the network directly with the glove by using the different program we wrote, this way is less convenient but has the advantage that you can actually see how the network behaves (for the hand gesture data and for the behavior between two gestures). You can see the evolution of the output according to the hand shape.

Training is the most painful step in neural networks because it takes a long time and because you still have to check if the Network is converging to the right point. By the way, time is crucial factor, the more time you spend to train networks or on the training data, the better your results can be.

Once you think the function found behave is appropriate enough for your purpose, you may need to correct it a little, translate, recenter, scale...

C. Architecture of the system

1. Using RPC for a distributed environment

An RPC system is the collection of software necessary to support remote procedure call programming and the necessary run-time services. An RPC system is a logical subset of a distributed computing environment (definition taken from [1])

We used RPC (Remote Procedure Calling) developed by Sun Microsystems because it allows us to use a distributed environment. With RPC, a client can execute procedures on other networked computers so it is possible to have access to more resources throughout a network. In the case of ATR's Virtual Teleconferencing system, resources are divided between several workstations, some are dedicated to a particular task. We used a distributed environment so that the procedures called should not take away all the power of a single machine.

Distributed environments allows also flexibility, since we can have clients running on any machine (a server and a client can even exist on the same machine), data can be shared by several programs without any change for the server. These

clients can use the data however they want. This is indeed a major tool in the UNIX programming environment.

RPC makes the client/server model more powerful and easier to program than low level socket interface. When combined with the ONC RPCGEN protocol, compiler, clients transparently make remote calls trough a local procedure call interface.

2. An overview of the network protocols

RPC works over TCP/IP protocol network. For such a network, the program and supporting network services have to follow these steps:

1. - Determine the sending and the receiving programs, which could be distributed among several machines,
2. - cut the data stream into datagrams,
3. - send the datagrams to the receiving program and
4. - reassemble the datagrams into a data stream.

To locate a machine, the network services use the internet address. Each destination on a network has a unique Internet address. It is a 32-bit number represented as four 8-bits seperate numbers, covering 0-255. For instance the internet address of the client machine is 133.186.25.133 but the file /etc/hosts maps the internet number with a common name, in this case the name is ciris23. For Ethernet networks, an address resolution protocol (ARP) located outside Internet protocols translates the Internet address into a Ethernet address.

Datagrams are the unit of data sent or received. Because TCP/IP is a connectionless technology, you don't send a data stream directly but instead you have to break the stream into datagrams encapsulated in, or surrounded by, headers and trailers, forming packets. The size of the datagram is typically 576 bytes for Ethernet.

To correlate TCP/IP with the OSI model (which came after), we could say that the Internet protocols span the data-link, network, transport, and process layers of the OSI models. RPC comes on top of the layer 4 (Transport), in the layer 5 (Session).

For our system we had to choose between TCP and UDP transport protocol. TCP transport protocol takes responsability for making sure the commands arrive at

destination. Also if the message is too large, TCP will split it into several datagrams, then will reassemble the message at the other end. TCP is a reliable transport so when a reply message is received from the server, it means that the procedure was executed only once.

Thus we shall use TCP transport if:

1. Reliability is paramount,
2. problems arise when remote procedures are executed more than once and
3. the size of the reply or the request exceed the size of one UDP packet (usually 8kbytes).

On the other hand, UDP (user datagram protocol) is an unreliable protocol, though, as TCP, it does include datagram checksums, it doesn't retransmit datagrams when an error occured and sends only a single datagram, so if your message is larger than the datagram's size UDP won't make any attempt to fragment the message or to reassemble it at the other end. Since it is not reliable, when you receive a reply from the server, you cannot be sure the procedure has been executed only once.

Thus we shall use UDP if:

1. Remote procedures can be executed more than once without any problems,
2. server request messages and client reply messages can each fit completely within one UDP packet, and
3. the server handles multiple clients. TCP servers require to keep the connections open, with the number of connections limited by operating system. The same holds true for a client communicating with multiple servers simultaneously.

But since we are going to build a server for each neural networks, and since the remote procedures can be executed as often as we like and since we shall work within a *local network*, so reliability is not such a problem, we shall use the UDP transport for our system.

Above the *session layer* is the *representation layer*. Actually, data sent by the computer should be machine-independent. That is why ONC RPC uses a single-canonical represention format as data representaion known as External Data Representation (XDR). RPC provides interfaces that converts usual type formats into XDR format, from these you can build your own data format and convert it into XDR.

The user Application lies then on top of the OSI model, in the layer 7 (application) (See figure 18). See the complete architecture on figure 19.

Currently, servers are physically running on the same machine, but on different processes, with RPC it doesn't change anything and it is even easier to use than the Interprocess communication (IPC) Tools provided by UNIX.

We may notice that the servers that send the position and orientation of the focal point are also clients of the server that sends data from the Cyberglove.

D. Trajectory analysis

1. Preamble

Once you have the position of the focal point, its trajectory may give you the information you need. So we must build a module in our system that could analyze the motion of the focal point and transform it into information concerning the transformation.

We assume here that the *speech recognition* system has been already implemented. It should be clear that our system is intended to work in association with natural language. One of the advantages of combining both is that you have some information about when the gesture starts, also you already know what kind of transformation the user wants to have.

For our purpose we basically need to recognize translation and rotation. They are basic gestures that can compose more complex gestures. Therefore, with translation, by combining data from both hands, you can also recognize the command gestures for stretching or squeezing. With rotation you can control an object's orientation and you could even recognize a twisting gesture.

2. Translation, stretching, squeezing

However we have to take into account the fact that, even in real life, the precision of the gestures is very low so we cannot completely rely on the data we receive. we have to limitate ourselves in direction and amplitude.

For translation, stretching, and squeezing, we limitate ourselves to the three axis X, Y, Z. So for the transformations, the system considers the motion:

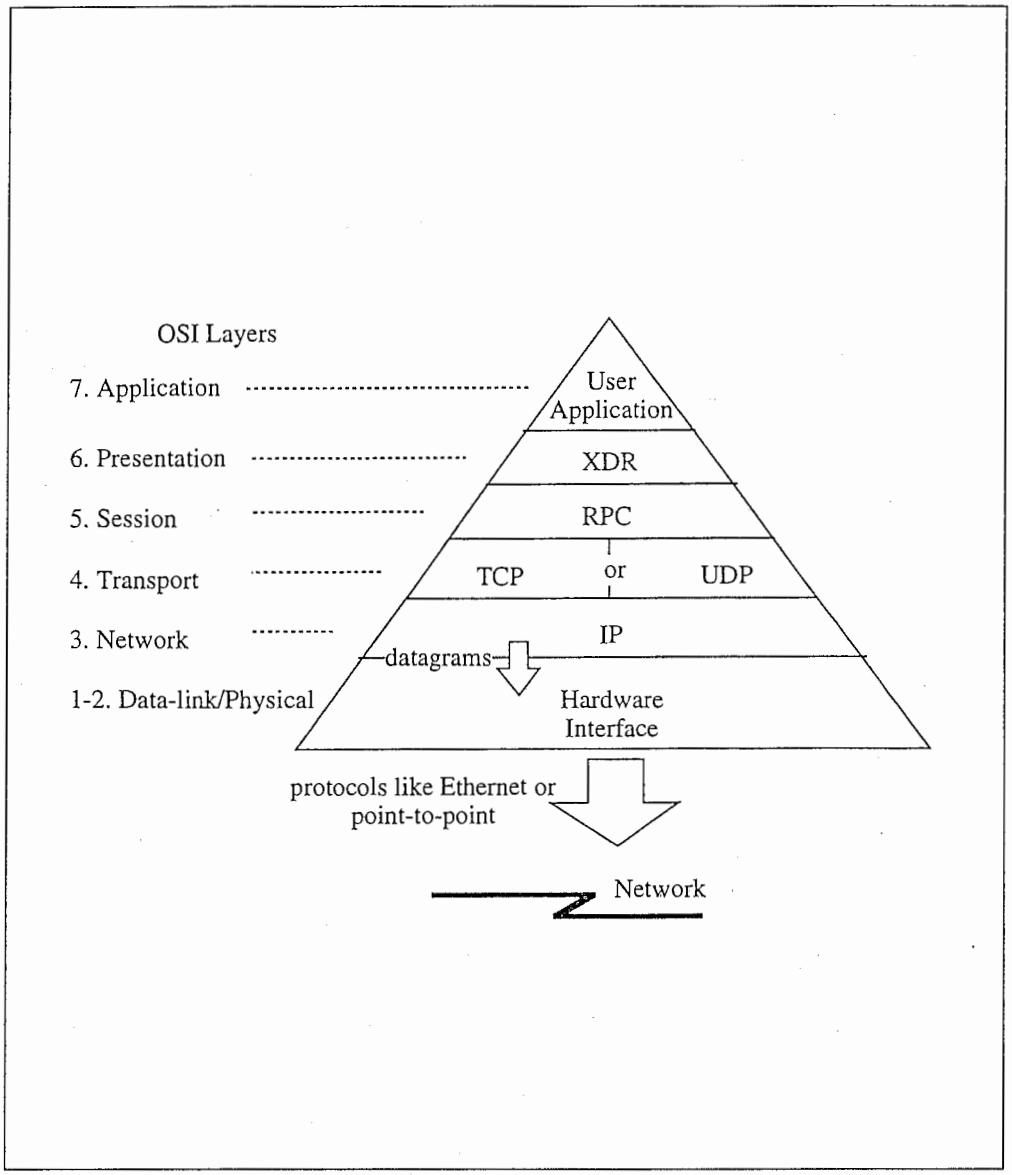


Figure 18: RPC within the OSI reference model.

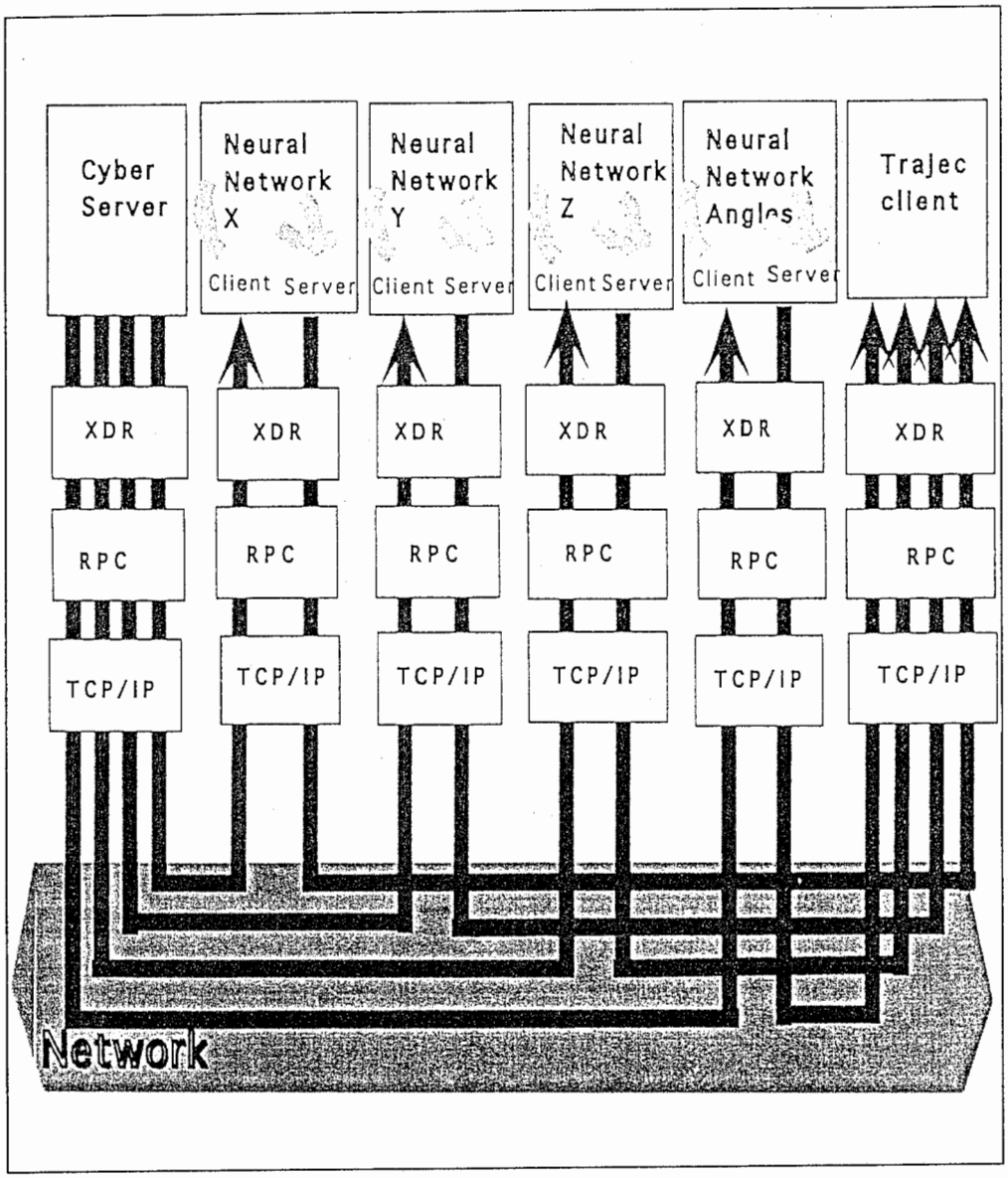


Figure 19: Architecture of the system.

- dorward,

- backward,

- leftward,

- rightward,

- upward and

- downward

Therefore the system will recognize which of these six directions the hand gesture is indicating and tell us what amplitude should be applied for the transformation on the object. Actually, given the amplitude of the movement, the amplitude of the transformation might be chosen according to the following.

1. The linear relation: the amplitude of the transformation and the gesture linearly correlated. This is the most common way to transforn the object. Still with this relation, we have to handle the problem of continuous enlargement.

2. Logarithmic relation: the amplitude of the transformation follows an exponential law.

3. Speed dependence: the speed of the motion gives the amplitude of the motion i.e. the faster you move the greater is the transformation. This relation starts from the assumption that, when you want to make a small transformation, you tend to move slower.

For the moment we just implemented the first relation. Future works on the subject should also deal with other relations. The algorithm is actually quite simple:

Let x, y, z be the coordinates for a hand position x_r, y_r, z_r and x_l, y_l, z_l
for the right and left hands respectively
At time t, calculate the variation of the three coordinates: dx, dy, dz
(respectively $d(x_r - x_l), d(y_r - y_l), d(z_r - z_l)$).
Calculate the maximum of the variations, then choose a direction for the translation.
If the transformation has already begun,

check if the direction is the
same as before.
if it is,
the amplitude is a function of
$dx^2 + dy^2 + dz^2$
else,
increase a counter and
do not apply any effect,
if the counter reaches a certain level,
end the transformation.

Do it until several variations are null or very small.

3. Rotation

We will also limit the axis of rotation to one of the three X-Y-Z axis. However, to determine the amplitude of the rotation and the axis around which the transformation, we shall use a more complex algorithm. We shall first construct a medium plane which is the closest to the points of the trajectories:

Let $M_0 ... M_n$ be the points of the trajectory described by the focal point. $(x_0, y_0, z_0) \ldots (x_n, y_n, z_n)$ their coordinates. First we shall calculate the axis around which the rotation occurs. To do that, we shall calculate the closest plane to the last points $M_{n-p}, M_{n-p+1}, \ldots M_n$, we assume that these p points are not on the same line.

Let P_p be a plane in the space. Its general equation is:

$$a_p x + b_p y + c_p z + d_p = 0, \forall x, y, z \in \Re$$

Let d_{ip} be the distance between M_i and P_p:

$$d_{ip} = \frac{|a_p x_i + b_p y_i + c_p z_i + d_p|}{\sqrt{a^2 + b^2 + c^2}}$$

P_p is the closest plane to $M_{n-p}, \ldots M_n$ if (a_p, b_p, c_p, d_p) minimize th following function F defined by:

$$\mathcal{F}(a_p, b_p, c_p, d_p) = \sum_{i=n-p}^{n} d_{ip}^2, \forall x$$

so (a_p, b_p, c_p, d_p) must verify the following equations:

$$\frac{d\mathcal{F}}{da}(a_p, b_p, c_p, d_p) = 0$$

$$\frac{d\mathcal{F}}{db}(a_p, b_p, c_p, d_p) = 0$$

$$\frac{d\mathcal{F}}{dc}(a_p, b_p, c_p, d_p) = 0$$

$$\frac{d\mathcal{F}}{dd}(a_p, b_p, c_p, d_p) = 0$$

The resolution of these equations should give (a_p, b_p, c_p, d_p) and so it is possible to define the plane P_p.

a_p, b_p, c_p, d_p are also the coordinates of the normal vector of the plane. Given this vector, we can choose the axis of the rotation. Then we should calculate the angular increasing during the motion.

Let $M_{n-2}, \ldots M_n$ be three consecutive points of the trajectory which are not on same straight line.

Let $I_{n-1} and I_{n-2}$ be the middle points of $M_{n-2}M_{n-1}$ and $M_{n-1}M_n$. To calculate the position of the center of the circle that passes by these three points, we calculate the plane defined by these points. On this plane,

Let O be the center of the circle, we have:

$$\overline{OI_{n-1}}.\overline{M_{n-2}M_{n-1}} = 0$$

$$\overline{OI_n}.\overline{M_{n-1}M_n} = 0$$

These equations define the position of O in the plane, so we can calculate the angle increasing by:

$$\alpha = \arctan \frac{OM_{n-2}}{M_{n-2}M_{n-1}} + \arctan \frac{OM_{n-1}}{M_{n-1}M_n}$$

This should be done only if O is at a certain range from M_n, if it is too close, it means that it may be an imprecision in the trajectory.

If $M_{n-2}, \ldots M_n$ are on the same straight line, we should still increase the angle of a small arbitrary amount, because of the possible imprecision of the trajectory.

VI. Experimental results

A. Detection of the focal point and its trajectory

To show how the focal point behave in a 3-D space, we built an application, using WorldToolkit (a library of virtual reality functions), that displays both the position of the wrist and the focal point (green sphere and red pyramid). Because of the behavior of neural networks, we had to filter the trajectory to avoid noise caused by singular points. The filter we programmed is the following. Let p_{nn} be the relative position given by the output of the neural networks. Let p_{final} and p_{for} be the final and former position of the focal point. The filter calculates p_{final} so that

$$p_{final} - p_{former} = \frac{1}{force} * (p_{nn} - p_{former})$$

Where $force$ is the force of the filter. This way, singular points have less influence on the position of the focal point, the counterpart is that the motion gets slower. Figures 20 21 22 show focal point trajectories with different filter forces.

In order to illustrate how the system works, let's consider a typical deictic gesture, when only the index finger is raised. The system recognizes this hand shape and moves the focal point to the finger's tip; We taught the system that, during this deictic gesture, this was where the attention was focused on. To show how the system can follow the focal point during this gesture, figure 23 shows an example of writing using this recognition system.

The system can recognize the trajectory of the focal point for many different gestures. For instance figure 24, 25 and 26 shows the trajectory of the focal point for various gestures that described the same concept, rotation.

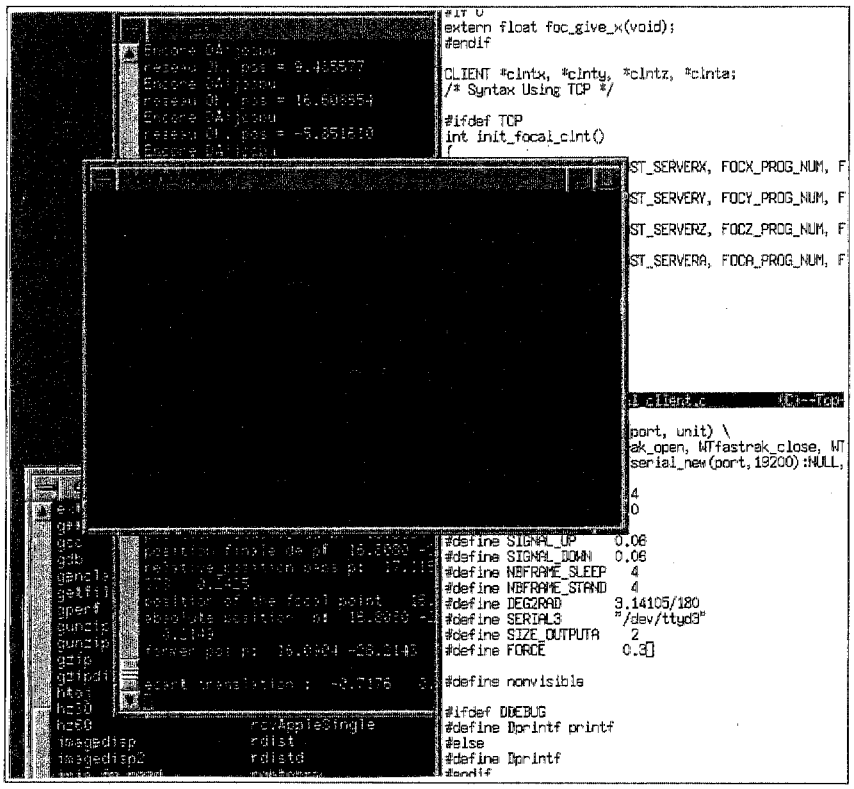


Figure 21: Trajectory with filtering force = 0.3.

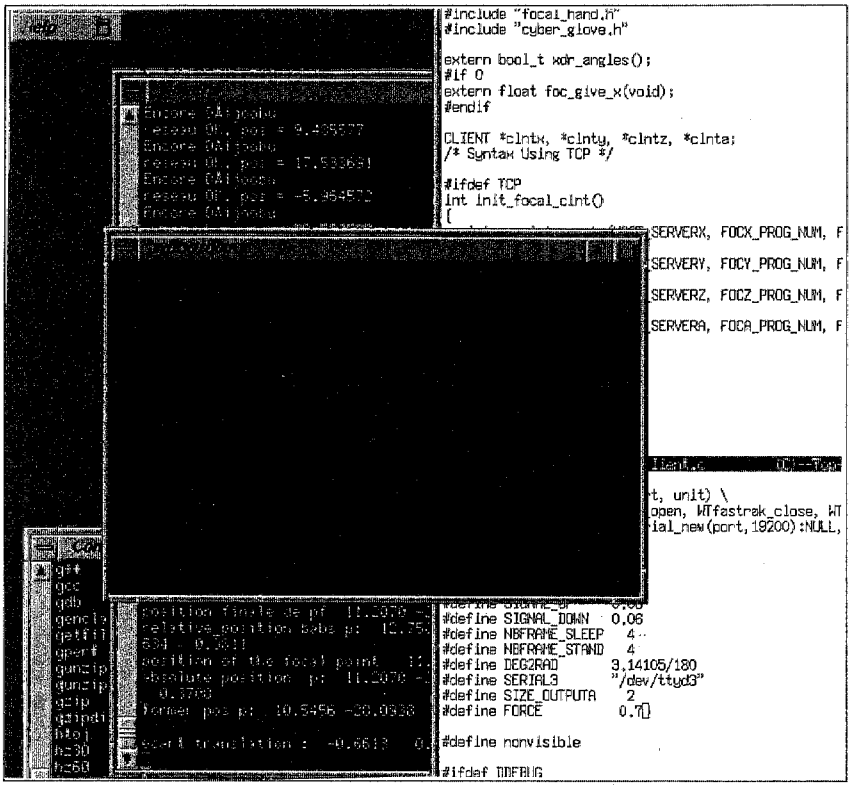


Figure 22: Trajectory with filtering force = 0.7.

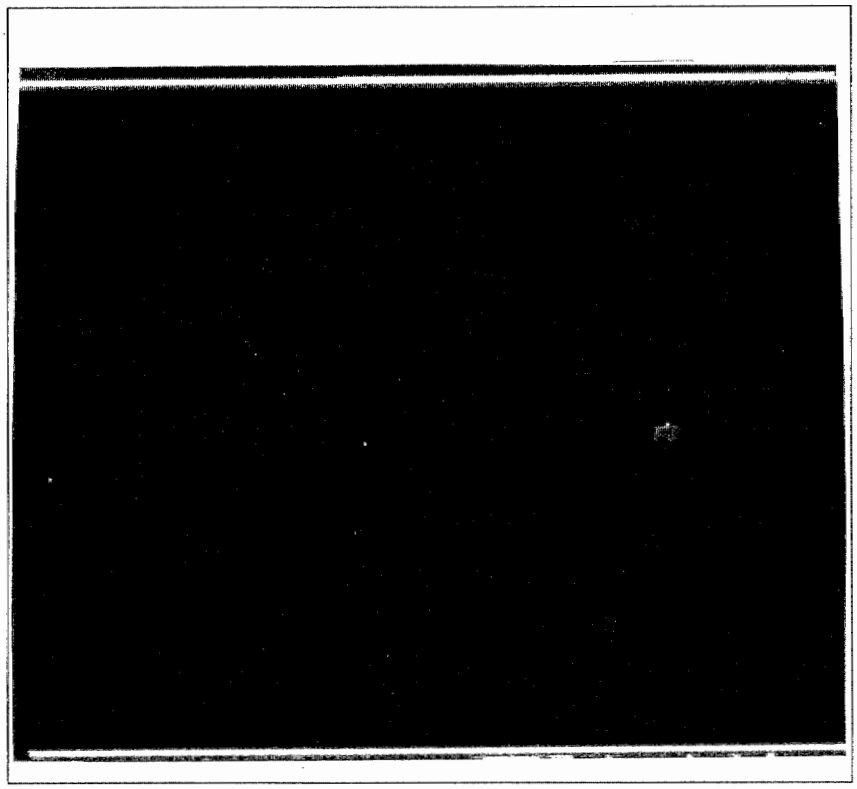


Figure 23: Writing with the finger tip.

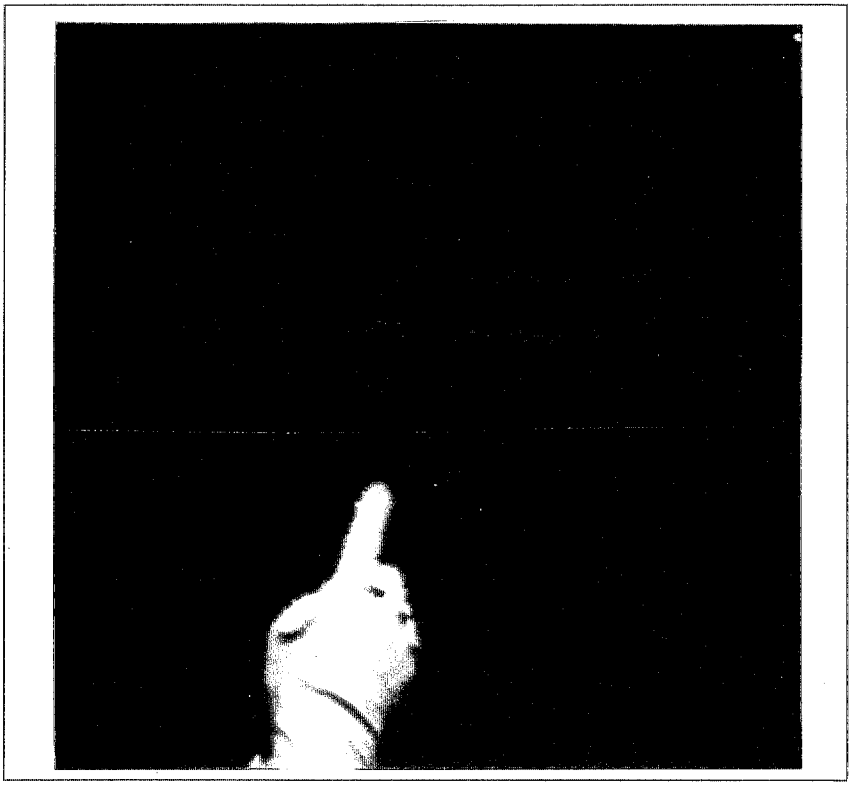


Figure 24: Trajectory and one gesture for a concept; rotation.

Figure 25: Trajectory and another gesture for the same concept; rotation.

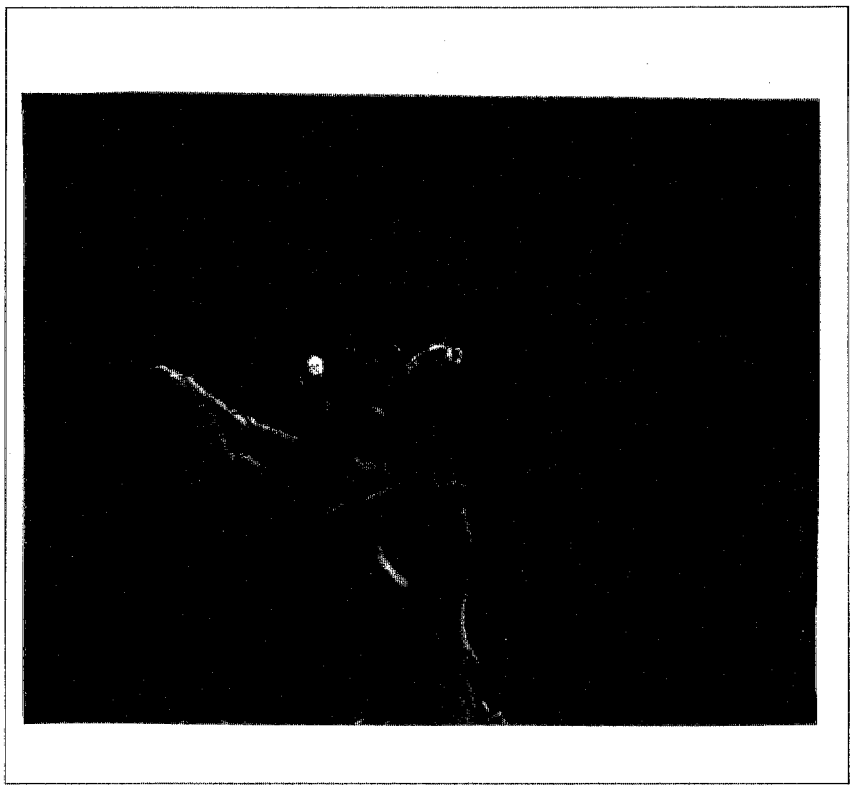


Figure 26: Trajectory and another gesture again for the same concept; rotation.

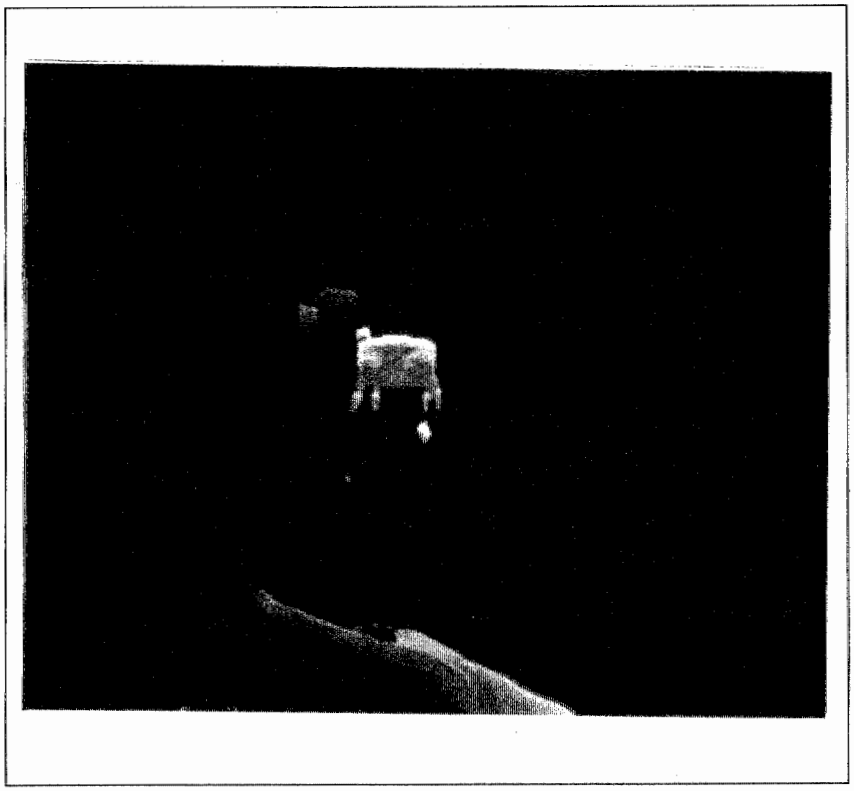


Figure 27: Trajectory analyzer: The virtual dog

B. Trajectory Analyzer: the virtual dog.

We built a system that can analyze trajectories describing a translation. The application has the shape of a virtual dog that can detect the meaning of natural hand gestures ("go there" or "seat down" for instance). The system hasn't been combined yet with speech recognition; to distinguish between different meaning, the system has a sequence of offsets to detect motion and transitions (see figure 27). Though the Fastrak is used only locally on the machine where WorldToolKit is running, the rest of the system works with RPC.

C. Advantages and Limitations

1. The advantages

    1. The advantage for such a method is that you don't depend as much on the shape of the hand as before. The problem of recognizing the gestures associated to a concept like rotating becomes tracktable because the shape of the hand has only an indirect influence on the recognition of the gesture.

    2. Because, as we saw, the position of the focal point depends only statically on the shape of the hand. You don't have to deal with time in the neural networks, so you don't have to use recurrent networks, which are difficult to train, or to integrate sequences in the gesture, which makes your networks become huge.

    3. Since the neural networks are very small, the necessary calculation time for the output is very small too. So it is possible to make them work in real time.

    4. Associated with Speech recognition, the system should allow the user to communicate with the machine a quite natural way. And also, since the system was built to work in a distributed environment, you can combine it with a classic sign recognition to have a full range of possibilities.

2. Limitations

    1. The main limitation of such a method is that you need to have a Trajectory analyzer. For rotation, you can use the algorithm described above, which is rather mathematical, or you can use Bezier curves to reconstruct the trajectory.

    2. neural networks don't make coffee. It means that you can't expect very precise results from the neural networks. You have to spend more time in the training stage and have a lot more of training data.

VII. Conclusion and Recommendations

The method we described in this paper introduces another approach for hand gesture recognition, it should be used as a complement to speech recognition as we use gestures as a complement for human-to-human communication. Considering the neural network part, using a bigger set of training hand gestures (the number of

weights in the Network should not be much bigger than the number of training data) should easily converge and enable us to use only one Network for the position of the focal point and increase the reliability. However, to increase precision, it might be necessary to use several output nodes for a single coordinate.

Because the one of our goals is to make it natural and intuitive for people to represent their mental images in 3D, this approach could take into consideration common sense knowledge of how we perceive things. To go further into a more intuitive human-to-human way of communication with help of computers, the first step is to understand how humans communicate with each other.

The application we built to recognize gestures using an indirect approach has a direct effect on a better understanding of human-to-human communication. This should be a subject of future research. We believe that perception is an essential part in Communication. Actually, the example of the focal point could be generalized into a general problem. How do we perceive people, doing gestures, talking, having different postures? With current technology available to us, for instance a gaze recognition system, we could hold an experiment and track the subject's gaze in order to understand better how he sees things as he talks and how it relates to listener.

VIII. Acknowledgements

I would like to thank President Nobuyoshi Terashima, Fumio Kishino, my supervisor, Yuri A. Tijerino, for their constant help and advice, and all the staff of ATR-CSRL for their support and help during my Internship, Roberto G. Lopez, for his patience and the Planning section, I couldn't have done anything without them. It has been indeed a great experience.

[1] J. Bloomer. *Power Programming with RPC.* O'Reilly & Associates, Inc., a nutshell handbook edition, 1992.

[2] J. Clark. Designing surfaces in 3-d. *Communications of the ACM*, 19(8):514–523, 1976.

[3] J. Darrell and A. Pentland. Space-time gestures. In *Proc. Conf. on Computer Vision and Pattern Recognition*, pages 335–340, New York, NY, June 15-17 1993.

[4] S. S. Felds. Glove-talk system. Technical report, University of Toronto, 1990.

[5] F. Kishino. Communication with realistic sensations through integrated multimedia environment. In *Proc. of Language and Vision Workshop(1st US-Japan Workshop on Integrated Systems in Multimedia Environments)*, pages 49–58, 1991.

[6] R. R. Leighton. *The Aspirin/MIGRAINES Neural Network Software, User's Manual*, 1992.

[7] G. Mouravapin and F. K. H. Takemura. Hand motion interpretation using neural networks. Technical Report 59, ATR Communication Systems Research Laboratories, 1991.

[8] A. P. Pentland. Perceptual organization and the representation of form. *Artificial Intelligence*, 28:293–331, 1986.

[9] A. P. Pentland. Thingworld. Technical Report 173, Vision and Modeling Group, The Media Lab, MIT, July 18 1990.

[10] J. Schlenzig, E. Hunter, and R. Jain. Recursive identification of gesture inputs using hidden makov models. In *IEEE Workshop on Applications of Computer Vision*, Sarasota, Florida, December 5-7 1994.

[11] Y. A. Tijerino, S. Abe, and F. Kishino. Modeling with 3-d shape ontologies and implicit functions in a virtual workspace. In *AID94 Workshop on reasoning with shapes in design.* Artificial Intelligence in Design, 1994.

[12] Y. A. Tijerino, S. Abe, T. Miyasato, and F. Kishino. What you say is what you see, –interactive generation, manipulation and modification of 3-d shapes based on verbal descriptions–. *Artificial Intelligence Review Journal*, 8(2), 1994.

[13] D. L. Vickers. *Sorcerer's Apprentice: Head-Mounted Display and Wand.* PhD thesis, Dept. of Computer Science, University of Utah, Salt Lake City, 1974.

[14] Virtual Technologies, Palo Alto, CA94306. *Cyberglove, User's Manual.*

[15] D. Weimer and S. K. Ganapathy. A synthetic visual environment with hand gesturing and voice input. In *CHI'89 Conf. Proc.*, pages 235–240, 1989.

[16] WorldToolkit. *WorlToolkit, User's Manual.*

[17] J. Yang, Y. Xu, and C. Chen. Gesture interface: Modeling and learning. In *Proc. International Conf. Robotics and Automation*, pages 1747–1752, San Diego, CA, May 8-13 1994.

IX. Appendix 1

Here is the listing of the file focalX.aspirin. This file is written in aspirin language. It describes a three layer Neural Network with 21 nodes at the hidden layer used to detect X coordinates. PdpNode3 is a sigmoidal node that has an output range of (-1,1). The name of the blackboxes for X, Y, Z, A, are all focal, so the name given by the Compiler will be the same. But since they are working seperately, it doesn't have any effect.

If you want to change the name of the Blackboxes, don't forget to change the name in the Recognitionxxxx files.

to execute Aspirin, type in the directory where Aspirin is,

bpmake focalX

then to train it : focalX -l -d yourdatafile.df -a xxx -f xxx etc...

Once Finished, save your Network.Finished file as RecognitionX

```
/*file focalX.aspirin*/

#define N_INPUTS 16
#define N_OUTPUTS 1
#define N_HIDDEN 21

DefineBlackBox focal
{
          OutputLayer-> Output
          InputSize->   N_INPUTS
  Components->
          {
                PdpNode3 Output [N_OUTPUTS]
                   {
                       InputsFrom-> Hidden_Layer
                    }
                PdpNode Hidden_Layer [N_HIDDEN]
                    {
                       InputsFrom-> $INPUTS
}
```

```
            }
}
```

here is the same for focalY, focalZ, focalA :

```
/*file focalY.aspirin*/

#define N_INPUTS 16
#define N_OUTPUTS 1
#define N_HIDDEN 25

DefineBlackBox focal
{
          OutputLayer-> Output
          InputSize->   N_INPUTS
  Components->
          {
                PdpNode3 Output [N_OUTPUTS]
                   {
                       InputsFrom-> Hidden_Layer
                    }
                PdpNode Hidden_Layer [N_HIDDEN]
                    {
                       InputsFrom-> $INPUTS
}
           }
}


/*file focalZ.aspirin*/
#define N_INPUTS 16
#define N_OUTPUTS 1
#define N_HIDDEN 23

DefineBlackBox focal
{
          OutputLayer-> Output
          InputSize->   N_INPUTS
```

```
  Components->
          {
                PdpNode3 Output [N_OUTPUTS]
                   {
                       InputsFrom-> Hidden_Layer
                    }
                PdpNode Hidden_Layer [N_HIDDEN]
                    {
                       InputsFrom-> $INPUTS
}
           }
}


/*file focalA.aspirin*/

#define N_INPUTS 16
#define N_OUTPUTS 2
#define N_HIDDEN 33

DefineBlackBox focal
{
          OutputLayer-> Output
          InputSize->   N_INPUTS
  Components->
          {
                PdpNode3 Output [N_OUTPUTS]
                   {
                       InputsFrom-> Hidden_Layer
                    }
                PdpNode Hidden_Layer [N_HIDDEN]
                    {
                       InputsFrom-> $INPUTS
}
           }
}
```

X. Appendix 2

Here is the file `focal_serverX.c` this is the server for the X coordinate

```
#include "focal_handX.h"
#include "cyber_glove.h"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <math.h>
#include <sys/time.h>

#define Dprintf
#define DEBUG

void *malloc();
/*declaration of some functions */
static void foc_svc_run();

static int foc_svc(register struct svc_req *rqstp, register SVCXPRT   *transp);

extern void network_initialize(char *filename, int verbose);

/*************************************************************
               returns the output of the network
*************************************************************/
float foc_give_x(void)
{
/*  float *input, *input2, output, *output2; */
  float fake[] = {1};
  float input[SIZE_INPUT];
  float input2[SIZE_INPUT];
  float *output2;
  float output = 0;

printf("fonction appellee\n");
```

```
  printf("reseau initialise\n");

  give_data(input); /*get the ouput of the cyber_server*/

  input_filter(input, input2); /*apply a filter to these in order to send them
 into the NN */

  network_initialize(NETWORK_NAMEX, 0);

  focal_set_input(input2); /*the inputs of the network are set to input2*/

  focal_propagate_forward(); /*You have to call this function to have a
       change in the output*/

  output2 = focal_get_output(); /*takes the output*/

  output = output_unfilterX(output2); /*transform it into centimeters*/

  fflush(stderr);
  fflush(stdout);

  Dprintf("donnee calculee %f\n", output);
  return(output);
}

void main(argc,argv)
int argc;
char *argv;
{
  int i;
  int cnt;
  double  st, lt, get_systime();
  register SVCXPRT  *transp;
  int  proc_term();
  float focal_x;

  printf("sortie du neurone\n");

/* Syntax using TCP */
```

```
/*  if ((transp = svctcp_create(RPC_ANYSOCK,0,0)) == NULL) {
*    fprintf(stderr, "Error : svctcp_create\n");
*    exit(-1);
*  }
*/

/* Syntax using UDP */

  if ((transp = svcudp_create(RPC_ANYSOCK)) == NULL) {
    fprintf(stderr, "Error : svcudp_create\n");
    exit(-1);
  }

  pmap_unset(FOCX_PROG_NUM, FOCX_PROG_VER);
/* Using TCP */

/*  if (!svc_register(transp,
*                    FOCX_PROG_NUM, FOCX_PROG_VER, foc_svc, IPPROTO_TCP)) {
*    fprintf(stderr, "Error : svc_register\n");
*    exit(-1);
*  }
*/

/* Using UDP */
  if (!svc_register(transp,
                    FOCX_PROG_NUM, FOCX_PROG_VER, foc_svc, IPPROTO_UDP)) {
    fprintf(stderr, "Error : svc_register\n");
    exit(-1);
  }


  printf("entree du neurone\n");

  foc_svc_run();
  fprintf(stderr, "Error : cg_svc_run returned!\n");
}

/* Runs the server client program (waits for a request)*/
void foc_svc_run()
{
```

```
  fd_set  readfds;
  int  dtbsz = getdtablesize();
  extern int  errno;

  while(1) {
    readfds = svc_fdset;
    switch(select(dtbsz, &readfds, NULL, NULL, NULL)) {
    case -1:
      if (errno == EINTR) continue;
      perror("select");
      return;
    case 0:
      break;
    default:
      svc_getreqset(&readfds);
    }
  }
}

/* Function called at a call */
int foc_svc(register struct svc_req *rqstp, register SVCXPRT   *transp)
{
  float focal_x;

printf("entree du serveur\n");

  switch(rqstp->rq_proc)
    {
    case NULLPROC:
      if (svc_sendreply(transp, xdr_void, 0) == 0) {
fprintf(stderr, "Error: rcp_service\n");
      return(1);
      }
      return;

      /* procedure called (specified by the client) */
    case FOCX_PROG:
      if (!svc_getargs(transp, xdr_void, 0))
{
  fprintf(stderr, "Error : svc_getargs\n");
```

```
  return;
}

      focal_x = foc_give_x(); /*takes the output of the NN*/

      Dprintf("donnee envoyee %f\n", focal_x);

      /*sends the data back to the clien (arguments must be passed by address */

      if (!svc_sendreply(transp, xdr_float, &focal_x))
{
  fprintf(stderr, "Error : svc_sendreply\n");
  return;
}

      return(0);
    default:
      svcerr_noproc(transp);
      return;
    }
}

double  get_systime()
{
  struct  timeval       Timeval;
  struct  timezone      Timezone;

  if (gettimeofday(&Timeval, &Timezone) == -1) {
    fprintf(stderr, "System call error 'gettimeofday' !!\n");
    exit(-1);
  }
  return((double)((long)Timeval.tv_sec +
                  (long)Timeval.tv_usec / (double)1000000.0));
}
```

XI. Appendix 3

focal_handX.h file

```
/*defines the host of the cyber server*/
#define HOST "ciris23"

/* defines the name of the file of the Neural Network */
#define NETWORK_NAMEX "RecognitionX"

/*defines the size of the input and output*/
#define SIZE_INPUT 16
#define SIZE_OUTPUT 1
#define SIZE_OUTPUTA 2

/*important*/
/*These are the codes to call the procedure and the program on a server"
#define FOCX_PROG_NUM     0x40000400
#define FOCX_PROG_VER     1
#define FOCX_PROG         1

int  repere;

float unfiltered[SIZE_OUTPUT];

/*Useful filter functions */

/* filter the input in order to put them into the Neural Net */
extern int input_filter(float input[SIZE_INPUT], float input2[SIZE_INPUT]);

/* the output of the Neural Net */
/* for X */
extern float output_unfilterX(float *output);

/* Useful Network functions */

extern void network_initialize(char *filename, int verbose);
```

```
extern void network_load(char *filename);

/* Useful Network X functions */

extern int focal_init_network();

extern void focal_propagate_forward();

extern float *focal_get_output();

extern void focal_set_target_output(float *fake);

extern void focal_set_backprop_counter(int x);

extern void focal_set_input(float *input);

/* Useful Cyber_glove functions */

extern int give_data(float donnees[SIZE_INPUT]);
```

XII. Appendix 4

cyber_give_data.c file gets the data from the cyber_server

```
#include "cyber_glove.h"
#include "focal_handX.h"
#include "focal_handY.h"
#include "focal_handZ.h"
#include "focal_handA.h"

#include <stdio.h>

char    *xdr_cyber(), *xdr_cyber_angle();

/*this function calls the cyber_Server so acts as a client*/

int give_data(float donnees[SIZE_INPUT])
{
  int  cyber_data[24];
  int  stat, cnt;
  char hname[10];

  static struct CYBERSTR cy_str;

  sprintf(hname, HOST);

/* UDP Protocol is used between the cyber glove and the client */

  initrpcudp(hname, CG_PROG_NUM, CG_PROG_VER);

  if ((stat = callrpc(hname, CG_PROG_NUM, CG_PROG_VER, CG_PROG_PR2,
      xdr_void, 0, xdr_cyber_angle, &cy_str)) != 0)
    {
      fprintf(stderr, "Not Responce %s Machine\n", hname);
    }
  donnees[1] = cy_str.joint_angle[0][1];
  donnees[2] = cy_str.joint_angle[1][0];
  donnees[3] = cy_str.joint_angle[1][1];
```

```
  donnees[4] = cy_str.joint_angle[2][0];
  donnees[5] = cy_str.joint_angle[2][1];
  donnees[6] = cy_str.joint_angle[3][0];
  donnees[7] = cy_str.joint_angle[3][1];
  donnees[8] = cy_str.joint_angle[4][0];
  donnees[9] = cy_str.joint_angle[4][1];
  donnees[10] = cy_str.joint_space[0];
  donnees[11] = cy_str.joint_space[1];
  donnees[12] = cy_str.joint_space[2];
  donnees[13] = cy_str.joint_space[3];
  donnees[15] = cy_str.wrist_angle[0];
  donnees[16] = cy_str.wrist_angle[1];
  exitrpcudp();
  return(1);
}

/*defines our own RPC functions*/

static struct sockaddr_in  server_addr_udp;
static struct timeval  calludp_timeout;
static CLIENT *client;

initrpcudp(host, pgnum, vernum)
     char  *host;
     int   pgnum, vernum;
{
  struct hostent  *hp;
  int  socket = RPC_ANYSOCK;

  if ((hp = gethostbyname(host)) == NULL) {
    fprintf(stderr, "Error : Can't get %s hostaddress\n", host);
    exit(-1);
  }
  bcopy(hp->h_addr, (caddr_t)&server_addr_udp.sin_addr, hp->h_length);
  server_addr_udp.sin_family = AF_INET;
  server_addr_udp.sin_port = 0;

  calludp_timeout.tv_sec = 3;
  calludp_timeout.tv_usec = 0;
```

```
  if ((client = clntudp_create(&server_addr_udp, pgnum, vernum,
       calludp_timeout, &socket)) == NULL) {
    perror("clntudp_create");
    return(-1);
  }

  return(1);
}

callrpcudp(pcnum, inproc, in, outproc, out)
     int   pcnum;
     char  *in, *out;
     xdrproc_t  inproc, outproc;
{
  enum clnt_stat clnt_stat;

  clnt_freeres(client, outproc, out);
  clnt_stat = clnt_call(client, pcnum,
inproc, in, outproc, out, calludp_timeout);

  return((int)clnt_stat);
}

exitrpcudp()
{
  clnt_destroy(client);

  return(1);
}
```

XIII. Appendix 5

defiltre.c treats the data coming in and out of the NN

```
#include "focal_hand.h"
#include <stdio.h>
#include <math.h>

#define RAD2DEG     180/3.14159

int input_filter(float input[SIZE_INPUT], float filtered[SIZE_INPUT])
  {

    filtered[0] = -(input[0]-240)/134;
    filtered[1] = -(input[1]-182)/94;
    filtered[2] = -(input[2]-214)/115;
    filtered[3] = -(input[3]-259)/95;
    filtered[4] = -(input[4]-207)/160;
    filtered[5] = -(input[5]-250)/185;
    filtered[6] = -(input[6]-206)/144;
    filtered[7] = -(input[7]-223)/167;
    filtered[8] = -(input[8]-233)/186;
    filtered[9] = -(input[9]-213)/142;
    filtered[10] = -(input[10]-84)/123;
    filtered[11] = -(input[11]-72)/146;
    filtered[12] = 0.5;
    filtered[13] = -(input[13]-29)/85;
    filtered[14] = -(input[14]+20)/130;
    filtered[15] = -(input[15]-7)/81;

    return(1);
  }

float sq(a)
     float a;
{
  return(a*a);
```

```
}

/* Unfilter the output of Xso that it matches the coordinates */
float output_unfilterX(output)
float *output;
{
  float unfiltered[SIZE_OUTPUT];

  unfiltered[0] = (float)(11*output[0]+5);

  printf("donnee filtree %f\n", unfiltered[0]);

  return(unfiltered[0]);
}


/* Unfilter the output of Y so that it matches the coordinates */
float output_unfilterY(output)
float *output;
{
  float unfiltered[SIZE_OUTPUT];

  unfiltered[0] = (float)(10.5*output[0]+7.5);

  printf("y filtree %f\n", unfiltered[0]);

  return(unfiltered[0]);
}

float foncZ(float a)
{
  return(a*a*a);
}

/* Unfilter the output of Z so that it matches the coordinates */
float output_unfilterZ(output)
float *output;
{
  float unfiltered[SIZE_OUTPUT];
```

```
  unfiltered[0] = (float)(15*output[0]+3);

/*You can use another fonction to unfilter the output of the Network */
/*  unfiltered[0] = fonc((15*output[0]-7)/10);
 */
  printf("z filtre %f\n", unfiltered[0]);

  return(unfiltered[0]);
}


/* Unfilter the output of A so that it matches the coordinates
The given value, does not return a value, everything is passed as an argument*/
int output_unfilterA(float *output, float unfiltered[SIZE_OUTPUTA])
{

  unfiltered[0] = (float)(135*output[0]+45);
  unfiltered[1] = (float)(67.5*output[1]+22.5);

  printf("first angle %f\n", unfiltered[0]);

  return(1);
}
```

XIV. Appendix 6

file `focal_serverA.c` (angles) differs from `focal_serverX.c` since everything is passed as an argument so no manipulation of pointers

```
#include "focal_handA.h"
#include "cyber_glove.h"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <math.h>
#include <sys/time.h>

#define Dprintf printf

extern bool_t xdr_angles();

static void foc_svc_run();

static int foc_svc(register struct svc_req *rqstp, register SVCXPRT   *transp);

int foc_give_a(float output[SIZE_OUTPUT]) /* output is an argument */
{
  float fake[] = {1};
  float input[SIZE_INPUT];
  float input2[SIZE_INPUT];
  float *output2;

  network_initialize(NETWORK_NAMEA, 0);

  give_data(input);

  input_filter(input, input2);

  focal_set_input(input2);

  focal_propagate_forward();
```

```
  output2 = focal_get_output();

  output_unfilterA(output2, output); /*passes output as an argument*/

  Dprintf("sortie du neurone\n");

  printf("sortie angles %f %f\n", output[0], output[1]);

  fflush(stderr);
  fflush(stdout);

#if 0
  if (input) free((char *)input);
  if (input2) free((char *)input2);
  if (output) free((char *)output);
  if (output2) free((char *)output2);
#endif

  return(1);
}

void main(argc,argv)
int argc;
char*argv;
{
  int i;
  int cnt;
  double  st, lt, get_systime();
  register SVCXPRT  *transp;
  int  proc_term();


/* Syntax using TCP */

/*  if ((transp = svctcp_create(RPC_ANYSOCK,0,0)) == NULL) {
*    fprintf(stderr, "Error : svctcp_create\n");
*    exit(-1);
*  }
*/
```

```
/* Syntax using UDP */

  if ((transp = svcudp_create(RPC_ANYSOCK)) == NULL) {
    fprintf(stderr, "Error : svcudp_create\n");
    exit(-1);
  }

  pmap_unset(FOCA_PROG_NUM, FOCA_PROG_VER);
/* Using TCP */

/*  if (!svc_register(transp,
*                    FOCAPROG_NUM, FOCA_PROG_VER, foc_svc, IPPROTO_TCP)) {
*    fprintf(stderr, "Error : svc_register\n");
*    exit(-1);
*  }
*/

/* Using UDP */
  if (!svc_register(transp,
                    FOCA_PROG_NUM, FOCA_PROG_VER, foc_svc, IPPROTO_UDP)) {
    fprintf(stderr, "Error : svc_register\n");
    exit(-1);
  }

  foc_svc_run();
  fprintf(stderr, "Error : cg_svc_run returned!\n");
}


void foc_svc_run()
{
  fd_set  readfds;
  int  dtbsz = getdtablesize();
  extern int  errno;

  while(1) {
    readfds = svc_fdset;
    switch(select(dtbsz, &readfds, NULL, NULL, NULL)) {
    case -1:
```

```
      if (errno == EINTR) continue;
      perror("select");
      return;
    case 0:
      break;
    default:
      svc_getreqset(&readfds);
    }
  }
}

foc_svc(rqstp, transp)
     register struct svc_req *rqstp;
     register SVCXPRT   *transp;
{
  float focal_a[SIZE_OUTPUTA];

  switch(rqstp->rq_proc)
    {
    case NULLPROC:
      if (svc_sendreply(transp, xdr_void, 0) == 0) {
fprintf(stderr, "Error: rcp_service\n");
      return(1);
      }
      return;
    case FOCA_PROG:
      if (!svc_getargs(transp, xdr_void, 0))
{
  fprintf(stderr, "Error : svc_getargs\n");
  return;
}

      if (!foc_give_a(focal_a))
{
  fprintf(stderr, "Error : could not get output of Network");
}
      /*everything is an argument, no use of malloc */
      printf("envoi angles %f %f\n", focal_a[0], focal_a[1]);

\* xdr_angles is a type defined in
```

```
      if (!svc_sendreply(transp, xdr_angles, focal_a))
{
  fprintf(stderr, "Error : svc_sendreply\n");
  return;
}
      return(0);
    default:
      svcerr_noproc(transp);
      return;
    }
}

double  get_systime()
{
  struct  timeval       Timeval;
  struct  timezone      Timezone;

  if (gettimeofday(&Timeval, &Timezone) == -1) {
    fprintf(stderr, "System call error 'gettimeofday' !!\n");
    exit(-1);
  }
  return((double)((long)Timeval.tv_sec +
                  (long)Timeval.tv_usec / (double)1000000.0));
}
```

XV. Appendix 7

this program defines different filters for external data representations

```
#include "cyber_glove.h"
#define SIZE_OUTPUTA 2

xdr_cyber(xdrsp, cdat)
     XDR *xdrsp;
     int cdat[];
{
  return(xdr_vector(xdrsp, cdat, 24, sizeof(int), xdr_int));
}

xdr_cyber_angle(xdrsp, cs)
     XDR *xdrsp;
     struct CYBERSTR  *cs;
{
  return(xdr_vector(xdrsp, cs->joint_angle, 10, sizeof(int), xdr_int) &&
 xdr_vector(xdrsp, cs->joint_space, 4, sizeof(int), xdr_int) &&
 xdr_vector(xdrsp, cs->wrist_angle, 2, sizeof(int), xdr_int));
}

xdr_calib_data(xdrsp, dat)
     XDR *xdrsp;
     int dat[];
{
  return(xdr_vector(xdrsp, dat, 34, sizeof(int), xdr_int));
}

/*this is the filter we used for the angles */
bool_t xdr_angles(XDR *xdrsp, float angles[SIZE_OUTPUTA])
{
  return(xdr_vector(xdrsp, angles, SIZE_OUTPUTA, sizeof(float), xdr_float));
}
```

XVI. Appendix 8

The client side

```
#include "focal_hand.h"
#include "cyber_glove.h"

extern bool_t xdr_angles();
#if 0
extern float foc_give_x(void);
#endif

CLIENT *clntx, *clnty, *clntz, *clnta;
/* Syntax Using TCP */

#ifdef TCP

\*creates the clients */
int init_focal_clnt()
{
  clntx = clnt_create(HOST_SERVERX, FOCX_PROG_NUM, FOCX_PROG_VER, "tcp");

  clnty = clnt_create(HOST_SERVERY, FOCY_PROG_NUM, FOCY_PROG_VER, "tcp");

  clntz = clnt_create(HOST_SERVERZ, FOCZ_PROG_NUM, FOCZ_PROG_VER, "tcp");

  clnta = clnt_create(HOST_SERVERA, FOCA_PROG_NUM, FOCA_PROG_VER, "tcp");

  return(1);
}

int destroy_focal_clnt()
{
  clnt_destroy(clntx);
  clnt_destroy(clnty);
  clnt_destroy(clntz);
  clnt_destroy(clnta);
```

```
 return(1);
}
/*gets the coordinates from the servers */
float focal_getx(void)
{
  float focal_x;
  enum clnt_stat stat;
  static struct timeval TIMEOUT = {25, 0};

#if 1
  printf("Encore DAijoobu\n");

  if (stat = clnt_call(clntx, FOCX_PROG, xdr_void, 0, xdr_float, &focal_x, TIMEOUT)
!=0)
    {
      fprintf(stderr, "Not ResponceX on %s Machine\n",HOST_SERVERX);
    }
  else
    {
      printf("reseau OK, pos = %f\n", focal_x );
    }
#endif
#if 0
  focal_x = foc_give_x();
#endif
  return(focal_x);

}

float focal_gety(void)
{
  float focal_y;
  enum clnt_stat stat;
  static struct timeval TIMEOUT = {25, 0};

  printf("Encore DAijoobu\n");

  if (stat = clnt_call(clnty, FOCY_PROG, xdr_void, 0, xdr_float, &focal_y, TIMEOUT)
!=0)
```

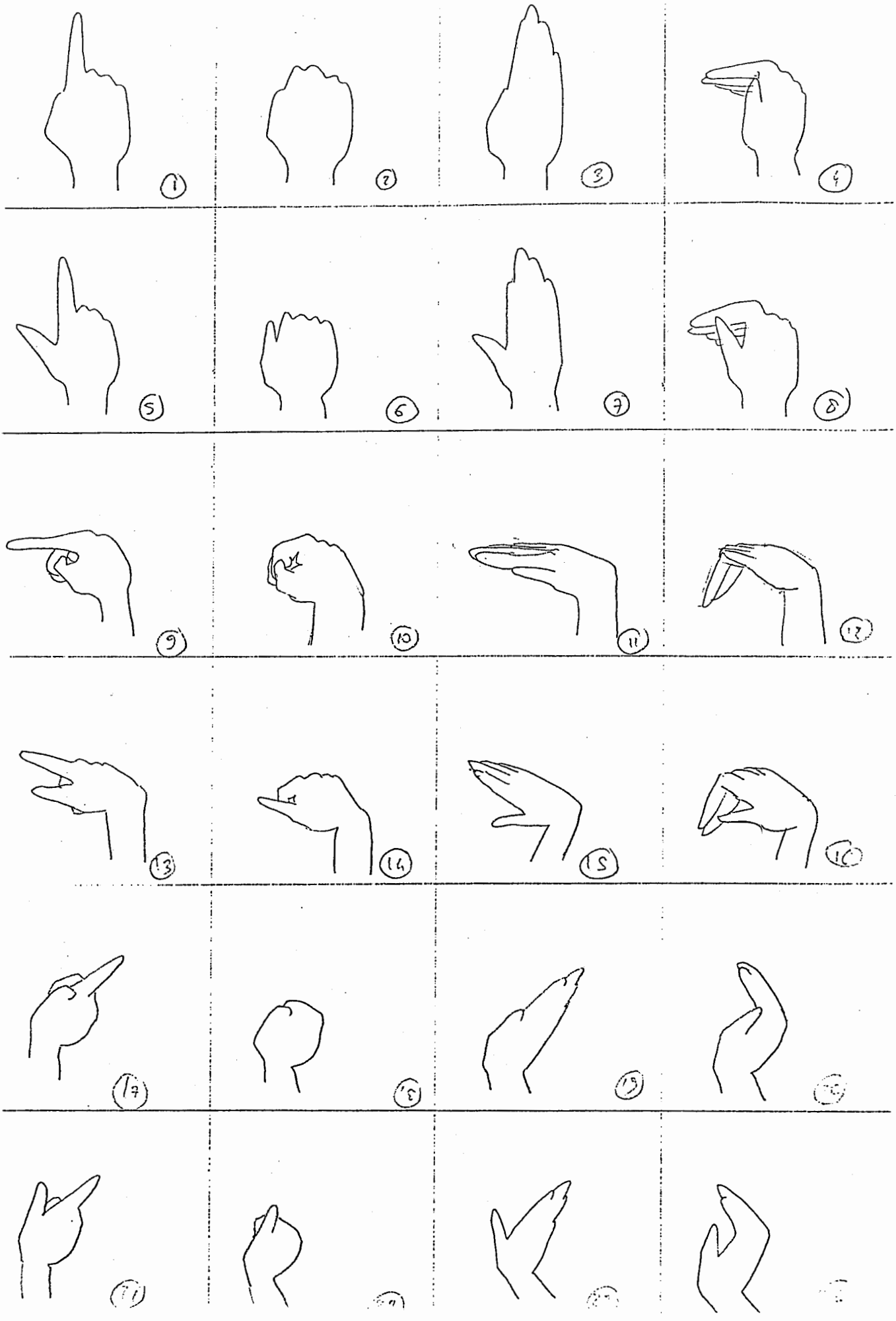
```
    {
      fprintf(stderr, "Not ResponceY on %s Machine\n",HOST_SERVERY);
    }
  else
    {
      printf("reseau OK, pos = %f\n", focal_y );
    }
  return(focal_y);
}

float focal_getz(void)
{
  float focal_z;
  enum clnt_stat stat;
  static struct timeval TIMEOUT = {25, 0};

  printf("Encore DAijoobu\n");

  if (stat = clnt_call(clntz, FOCZ_PROG, xdr_void, 0, xdr_float, &focal_z, TIMEOUT
!=0)
    {
      fprintf(stderr, "Not ResponceZ on %s Machine\n",HOST_SERVERZ);
    }
  else
    {
      printf("reseau OK, pos = %f\n", focal_z );
    }
  return(focal_z);
}

int focal_geta(float focal_a[SIZE_OUTPUTA])
{
  enum clnt_stat stat;
  static struct timeval TIMEOUT = {25, 0};

  printf("Encore DAijoobu\n");

  if (stat = clnt_call(clnta, FOCA_PROG, xdr_void, 0, xdr_angles, focal_a, TIMEOUT
!=0)
    {
```

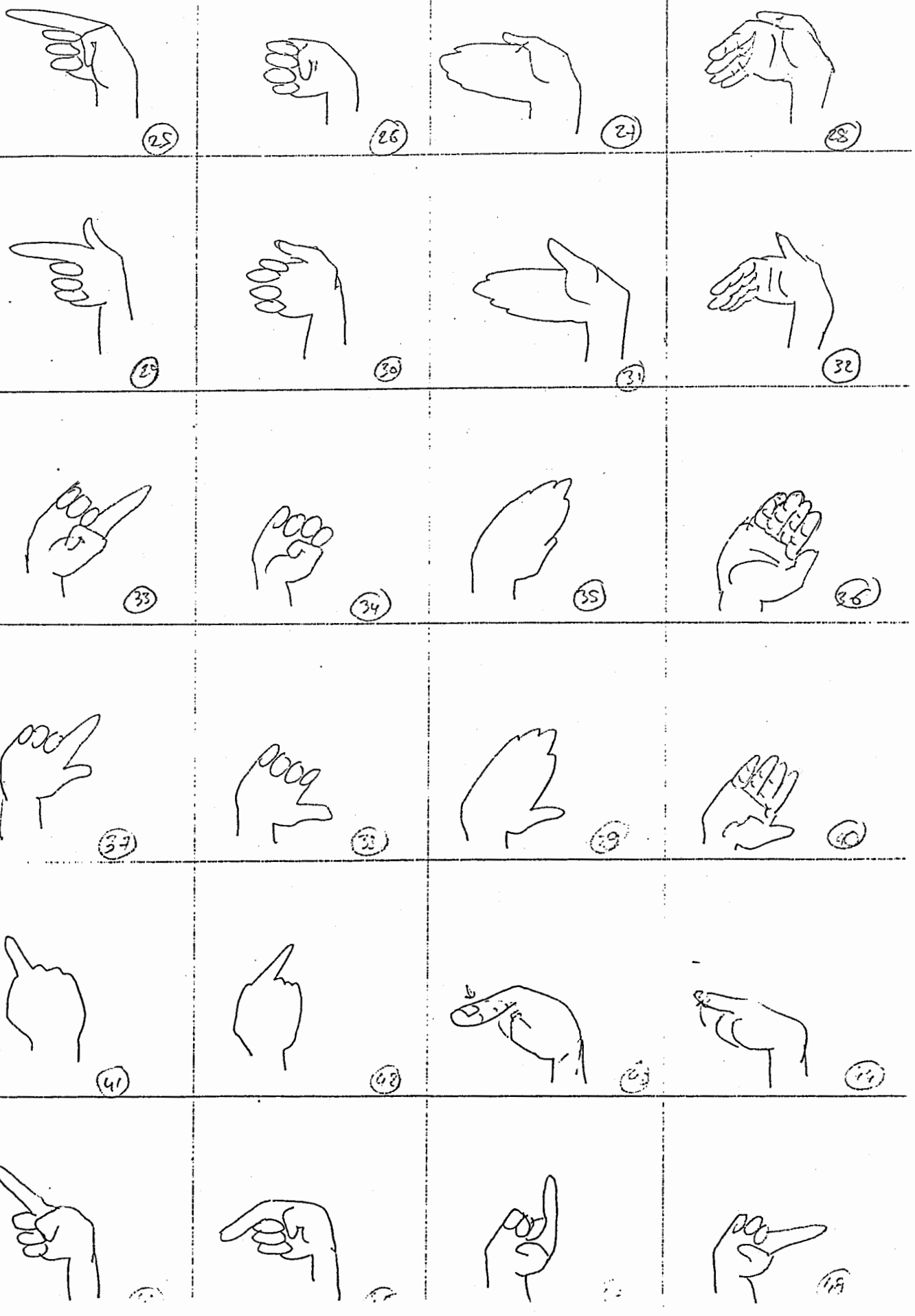
```
      fprintf(stderr, "Not ResponceA on %s Machine\n",HOST_SERVERA);
    }
  else
    {
      printf("reseau OK, pos = %f\n", focal_a[0] );
    }

  return(1);
}
#endif

/*Syntax using UDP*/

#ifdef UDP
float focal_getx(void)
{
  int stat;
  float focal_x;

  initrpcudp(HOST_SERVERX, FOCX_PROG_NUM, FOCX_PROG_VER);

  if ((stat = callrpc(HOST_SERVERX, FOCX_PROG_NUM, FOCX_PROG_VER,
                      FOCX_PROG, xdr_void, 0, xdr_float, &focal_x))!=0)
    {
      fprintf(stderr, "Not Responce on %s Machine\n",HOST_SERVER);
    }
  return(focal_x);
}

float focal_gety(void)
{
  int stat;
  float focal_y;

  initrpcudp(HOST_SERVERY, FOCY_PROG_NUM, FOCY_PROG_VER);

  if ((stat = callrpc(HOST_SERVERY, FOCY_PROG_NUM, FOCY_PROG_VER,
                      FOCY_PROG, xdr_void, 0, xdr_float, &focal_y))!=0)
    {
      fprintf(stderr, "Not Responce on %s Machine\n",HOST_SERVER);
```

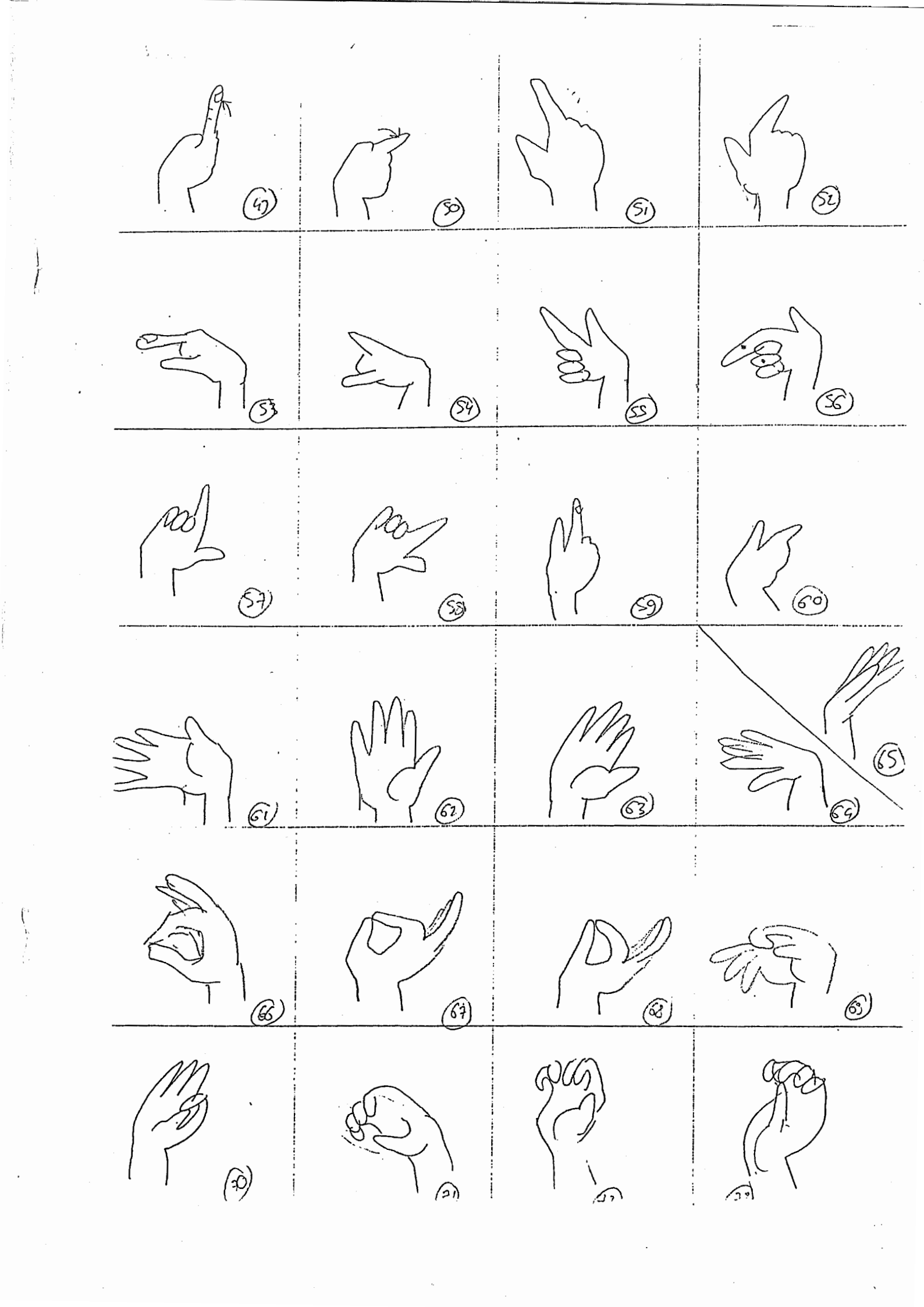
```
    }
  return(focal_y);
}

float focal_getz(void)
{
  int stat;
  float focal_z;

  initrpcudp(HOST_SERVERZ, FOCZ_PROG_NUM, FOCZ_PROG_VER);

  if ((stat = callrpc(HOST_SERVERZ, FOCZ_PROG_NUM, FOCZ_PROG_VER,
                      FOCZ_PROG, xdr_void, 0, xdr_float, &focal_z))!=0)
    {
      fprintf(stderr, "Not Responce on %s Machine\n",HOST_SERVER);
    }
  return(focal_z);
}

int focal_geta(float focal_a[SIZE_OUTPUTA])
{
  int stat;
  float focal_a[];

  initrpcudp(HOST_SERVERA, FOCA_PROG_NUM, FOCA_PROG_VER);

  if ((stat = callrpc(HOST_SERVERA, FOCA_PROG_NUM, FOCA_PROG_VER,
                      FOCX_PROG, xdr_void, 0, xdr_angles, focal_a))!=0)
    {
      fprintf(stderr, "Not Responce on %s Machine\n",HOST_SERVER);
    }
  return(focal_a);
}
#endif
```

XVII. ANNEX

Hand Shapes used for training data

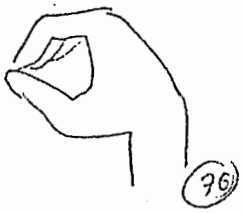

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21


25
26
24
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45


49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70


74


75


76


77

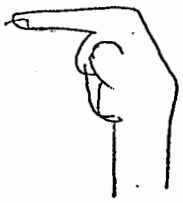

78


79

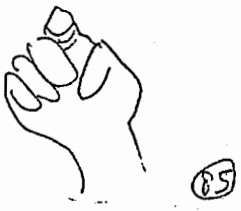

80


81


82


83


84


85


86


87


88


89


90


91


92