ＡＴＲテクニカルレポート表紙

〔非公開〕

TR－C－0059

HAND MOTION INTERPRETATION USING NEURAL NETWORKS

グナセゲラン　ムルヴァパン
Gounassegarin　Mourouvapin

竹村　治雄　　岸野　文郎
Haruo Takemura　　Fumio Kishino

１９９１．１．３０

ＡＴＲ通信システム研究所

HAND MOTION INTERPRETATION USING NEURAL NETWORKS

30 / 1 / 1991

Gounasségarin MOUROUVAPIN
Haruo TAKEMURA
Fumio KISHINO

ATR Communication Systems Research Laboratories
Artificial Intelligence Department

CONTENTS

ABSTRACT

This report describes a method of interpreting hand gestures. One of the applications could be manipulation of graphical objects on a computer screen. The devices used to detect hand and fingers position are the DataGlove (see ref [1]), and the 3SPACE Isotrak (see ref [2]). The basis of this method is neural networks. We are interested only in translation, rotation and scaling operations, and we suppose only fingers are used to scale objects. Data given by the Tracker (coordinates and angles) are computed for translation and rotation. Those provided by the Glove are the inputs of the network that calculates a parameter defining the scaling operation (enlargement or shrinking). The use of a simple neural network turned out to be quite limited to recognize some complex gestures, such as continuous growth gesture. We added recurrent connections in order to give the network memory and some power of prediction. Experimental results show that recurrent networks give the best responses.

I. INTRODUCTION

Human interface research such as recognition and understanding of human movements has seen the development of 3 dimensional gesture recognition devices such as the Sensor Frame, the Glove,...

Several experiments were conducted in hand motion interpretation (*for example ref [3]*), and one of the conclusions is that people prefer to use both speech and gestures to move objects displayed on the screen. However, we describe in this report a method where the only communication mode is gestures. The basis of this method is neural networks. These kind of networks have already been used to detect hand shapes (*see ref [4]*). Here, hand motion is detected.

We apply this method to manipulating simple graphical objects (2D objects) displayed on a computer screen.

We limit the operations to three basic ones : translation, rotation and scaling, and we suppose that for scaling operations, only fingers are used and hand is used for rotation and translation (*see figure 1 where these elementary operations are shown*).

These limitations are based on the fact that the DataGlove is not very sophisticated (*see next chapter*).

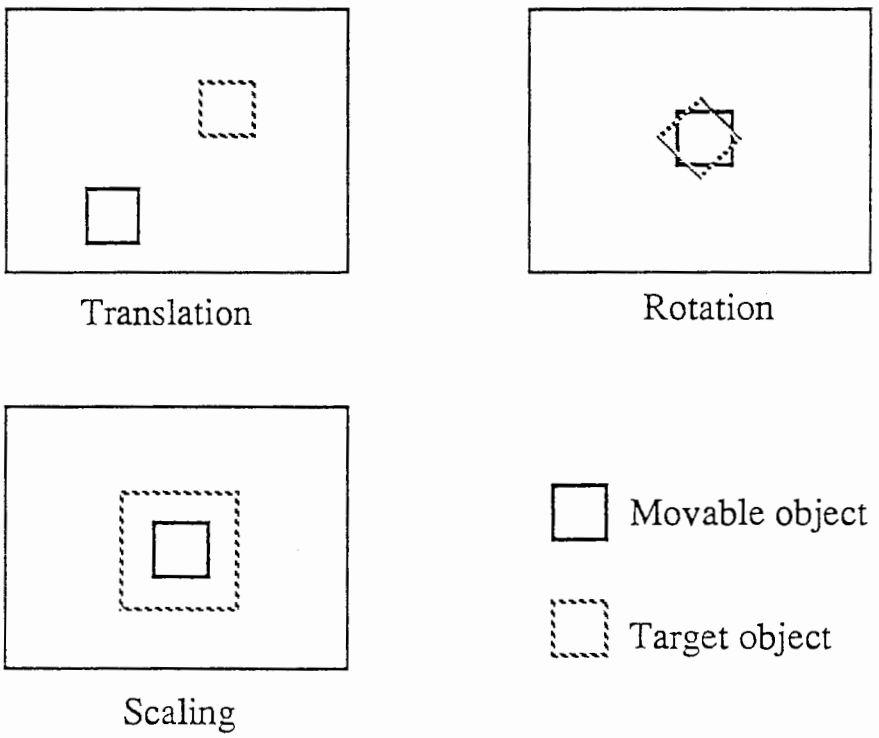


Figure 1 : Elementary operations.

II. THE DEVICES

2.1. DATAGLOVE AND 3SPACE TRACKER

The VPL Research DataGlove Model 2

The VPL Research DataGlove Model 2 allows for the translation of hand gestures into virtually any form of control signal. The DataGlove measures and calibrates movements in the finger joints while monitoring the hand's position and orientation in three-dimensional space. These calibrated measurements may be passed on to a host computer for conditioning into any required format.

The human hand itself is a very complex device, and its gestures can't be accurately measured by any single sensor technology. Thus, the DataGlove Model 2 employs two sensors technologies to measure gestures and absolute positioning of the hand :

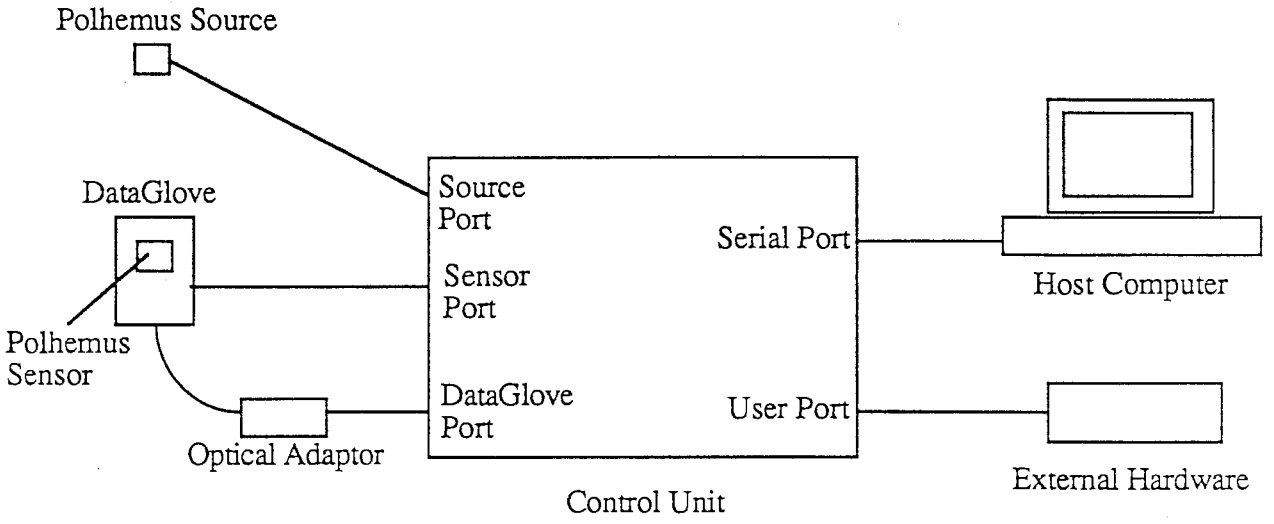


Figure 2 : Block diagram of the system.

Flex Sensors

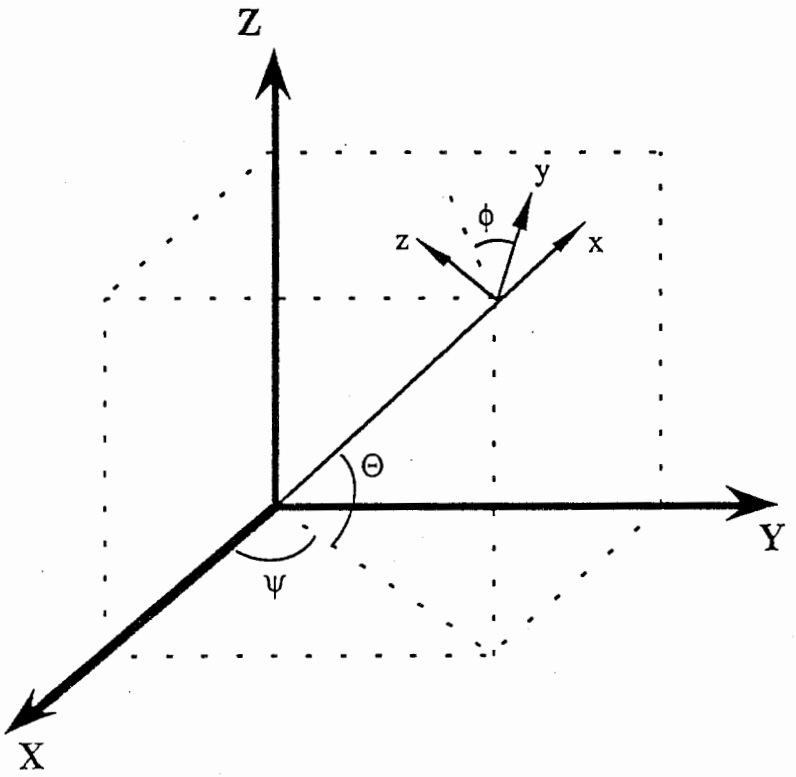
Flexing of the joints is measured by optical fibers which, when bent, attenuate light passing through them. Ten optical fibers run across the DataGlove to the inner and outer joints of the thumbs and fingers, and terminate at an optical adaptor containing the light sources for the fibers and the phototransistors which convert the attenuated light levels into analog voltages. These voltages are fed to the Control Unit via a 25-pin connector.

Polhemus Sensor

The DataGlove's absolute position in space is sensed by a Polhemus 3Space Isotrak sensor system. The 3Space system consists of two components :

- A "source", which emits electromagnetic fields. In the most common configuration, the source is mounted in front of the user.

- A "sensor", which responds to the source's fields according to its position relative to the source. The sensor feeds the Control Unit with six analogs signals, which describe the sensor's coordinates (x, y and z) and orientation (azimuth (Ψ), elevation (Θ) and roll (Φ): Euler angles) (*see figure 3*). The source is often the origin of the reference.



X, Y, Z : Alignment Reference Frame (reference related to the source).
x, y, z : Rotated Stylus or Sensor Coordinate Frame
Ψ : Azimuth
Θ : Elevation
Φ : Roll

Figure 3 : Euler angles.

(X, Y, Z) reference can be fixed as we like. However, the orientation of (x, y, z) depends on the sensor and is not subject to modification.

2.2 DATA FORMAT

The DataGlove, presented above, provides 10 integers related to the bending of each finger joints (2 for each finger) :

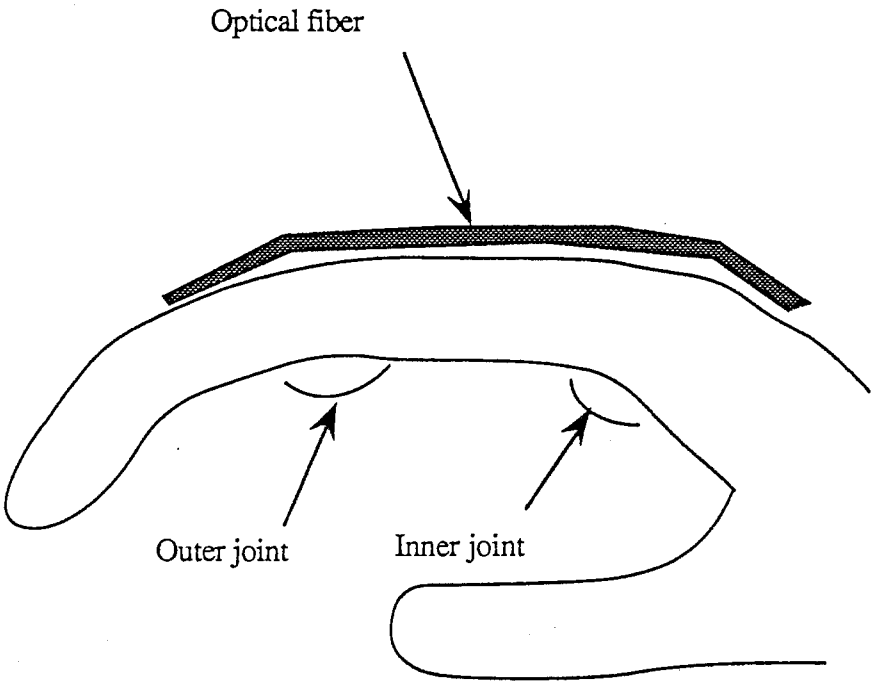


Figure 4 : Optical fiber along one finger.

These integers grow when opening the hand, because more light is then passing through the fibers. So when a finger is straight, the value related to it is maximum.

At any time, you can get the data from the Glove and from the 3Space Isotrak, which gives 6 floating point numbers : x, y, z and Euler angles. The format is the following :

time a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} x y z Ψ Θ Φ

a_1, a_2 : thumb inner and outer joints
a_3, a_4 : index inner and outer joints
a_5, a_6 : middle inner and outer joints
a_7, a_8 : ring inner and outer joints
a_9, a_{10} : pinky inner and outer joints

III. USING NEURAL NETWORKS

3.1. INTRODUCTION

The problem is, given the data (provided by the 2 devices), how to recognize movements of hand and fingers, and what the parameters to move objects on the screen should be.

As we are interested in motion, it is obviously the evolution of data that should be studied, not their absolute values.

To move an object, at least 3 parameters are required : translation vector, angle of rotation and a scaling factor.

The first two parameters are easy to compute using the 3Space Isotrak: just calculate the difference between two consecutive samples.

As we supposed that only fingers are used to enlarge or shrink, the scaling factor should be computed from the data given by the Glove. The problem is that there is no accurate and simple relation between the integers and the bending of the fingers. For ambiguous and complex finger motions, a simple equation is not sufficient (such an equation should at least have 10 variables). That is why we use a neural network, and its inputs will be the variation of the 10 integers. Its output will be the scaling factor, which is called *'growth'*.

3.2. A CLASSICAL NEURAL NETWORK

The first network implemented (in the C programming language) is a classical three-layer (perceptron) network with feedforward connections. A backpropagation learning method is used to train the network. Let's present this algorithm :

$A = (a_i)_i$ is the input pattern.
$C = (c_i)_i$ is the computed pattern
$T = (t_k)_k$ is the target pattern (desired output)
$W = (w_{ij})_{i,j}$ is the set of connections (or links) between F_B and F_C layers.
$V = (v_{hi})_{h,i}$ is the set of connections between F_A and F_B layers.
Each unit b_i of layer F_B has a threshold Θ_i.
Each unit c_j of layer F_C has a threshold Γ_j.

PE stands for Processing Element.

Using notations shown in *figure 5*, here is the backpropagation algorithm, called the *Vanilla backpropagation algorithm (see ref [6]):*

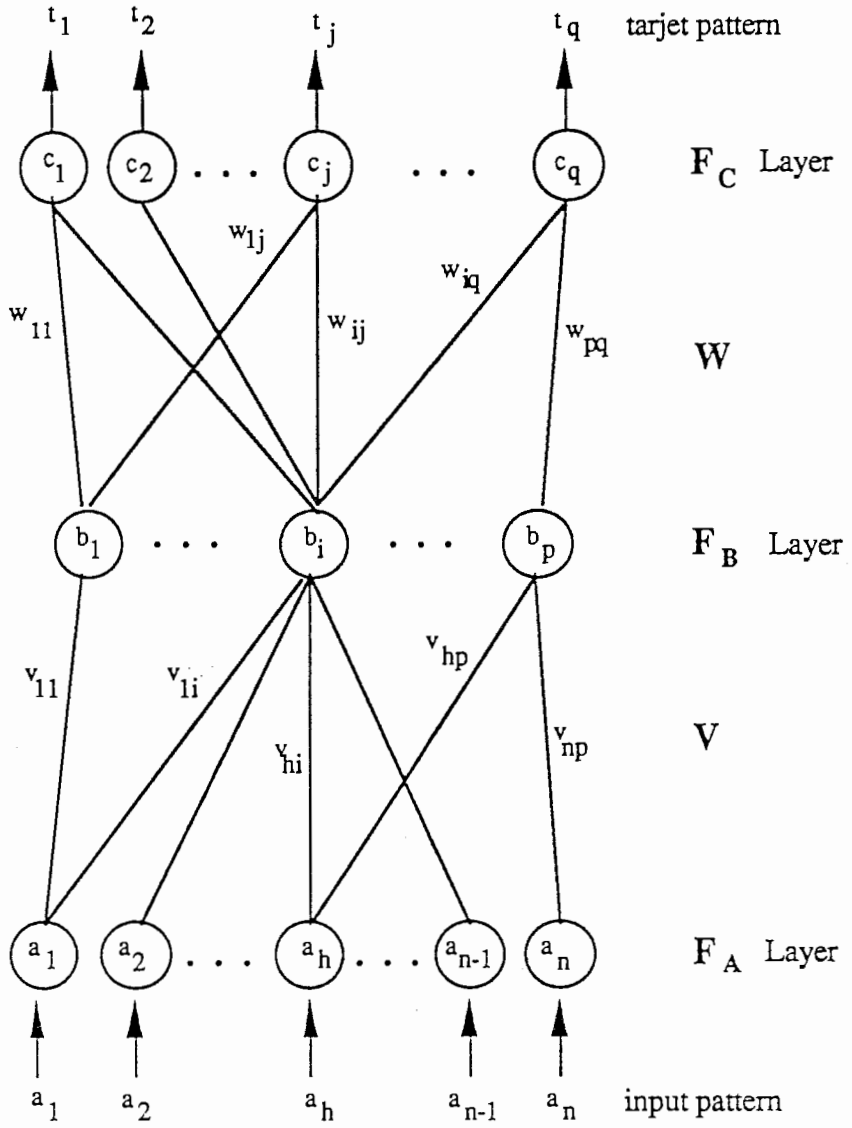


Figure 5 : Topology of elementary backpropagation feedforward recall network.

1. Assign random values in the range [-1, +1] to all the F_A to F_B inter-layer connections v_{ij}, to all the F_B to F_C inter-layer connections w_{ij}, to each F_B PE threshold Θ_i, and to each F_C PE threshold Γ_j.

2. For each pattern pair (A,T), do the following :

a. Transfer A's value to the F_A PEs, filter the F_A PE activations through V and calculate the new F_B PE activations using the equation :

$$b_i = f\left(\sum_{h=1}^{n} a_h v_{hi} + \Theta_i\right)$$

for all i = 1,..,p, where b_i is the activation value of the *ith* F_B PE, and f() is the logistic sigmoid threshold function $f(x) = (1 + e^{-x})^{-1}$. (Sigmoid functions have this form : $f(x) = A\ (1 - e^{-k_1 x})/(1 + e^{-k_2 x})$, where A, k_1 and k_2 are three positive constants).

b. Filter the F_B activations through W to F_C using the equation :

$$c_j = f\left(\sum_{i=1}^{p} b_i w_{ij} + \Gamma_j\right)$$

for all j = 1,..,q, where c_j is the computed output of the *jth* F_C PE.

c. Compute the error between the computed and desired output using the equation :

$$d_j = c_j(1-c_j)(t_j-c_j)$$

for all j = 1,..,q.

d. Calculate the error of each F_B PE relative to each d_j with the equation :

$$e_i = b_i(1-b_i)\sum_{j=1}^{q} w_{ij}d_j$$

for all i = 1,..,p.

e. Adjust the F_B to F_C connection weights

$$\Delta w_{ij} = \alpha b_i d_j$$

for all i = 1,..,p and all j = 1,..,q. Δw_{ij} is the amount of change made to the connection from the *ith* F_B to the *jth* F_C PE, and α is a positive constant called the *learning rate.*

f. Adjust the F_C thresholds

$$\Delta\Gamma_j = \alpha d_j$$

for all j = 1,..,q, where $\Delta\Gamma_j$ is the amount of change to the *jth* F_C PE's threshold value.

g. Adjust the F_A to F_B connections

$$\Delta v_{hi} = \beta a_h e_i$$

for all h = 1,..,n. β is like α a positive constant called *learning rate.* (we can take $\beta = \alpha$).

h. Adjust the F_B thresholds

$$\Delta\Theta_i = \beta e_i$$

for all i = 1,..,n.

3. Repeat step (2) until the error correction value d_j, for each j=1,..,p , and for each pattern A is either sufficiently low or zero.

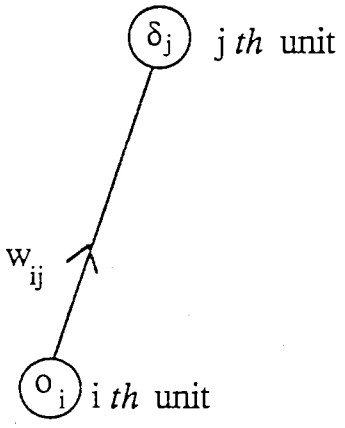
The learning rate and the momentum :

If we let

$$E_k = 1/2 \sum_j (t_j - c_j)^2 \qquad \text{(eq. 1)}$$

be the measure of the error on input/output pattern k, then theoretically the learning procedure requires that the change in weight be proportional to $\partial E_k / \partial w$. (*see Appendix 1 : the Generalized Delta Rule*). True gradiant descent requires that infinitesimal steps be taken. The constant of proportionality is the *learning rate* in the procedure. The larger this constant, the larger the changes in the weights (*see algorithm described above*). For practical purposes, we choose a learning rate that is as large as possible without leading to oscillation. This offers the most rapid learning. One way to increase the learning rate without leading to oscillation is to modify the rule to include a *momentum* term. This can be accomplished by the following rule :

$$\Delta w_{ij}(n+1) = \alpha \delta_j o_i + \eta \Delta w_{ij}(n)$$



δ_j and o_i are activations or outputs of units

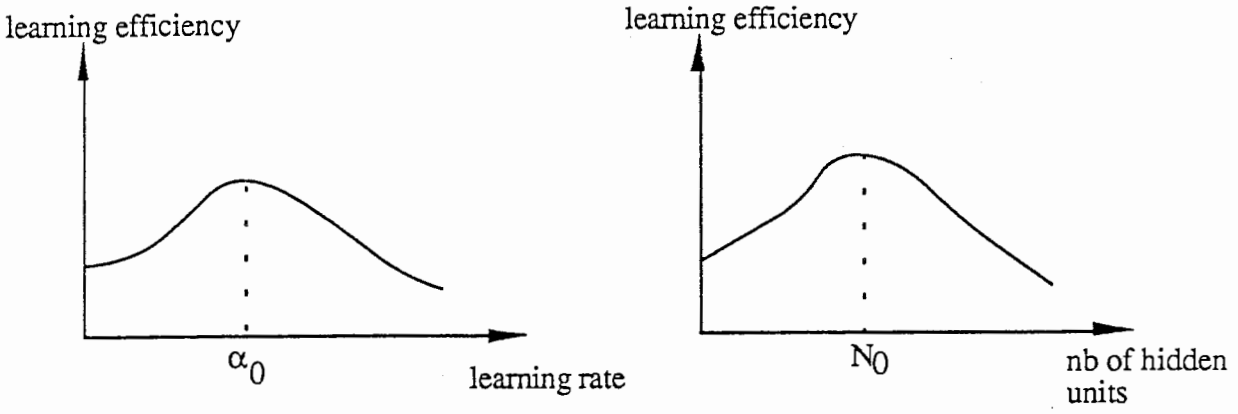
$\left.\begin{matrix}\delta_j = d_j \\ o_i = b_i\end{matrix}\right\}$ if it's a hidden unit

$\left.\begin{matrix}\delta_j = e_j \\ o_i = a_i\end{matrix}\right\}$ if it's an input unit

where the subscript n indexes the presentation number of patterns, α is the learning rate and η is a constant which determines the effect of past weight changes on the current direction of movement in weight space. This provides a kind of momentum in weight space. (It's actually what is done in our program).

Different parameters such as the learning rate and the number of hidden units, should be calculated to optimize the learning. But there is

no theory on how to compute those values. We just know that variation of the learning efficiency is as illustrated in the following graphs :



Only by experiments can α_0 and N_0 be estimated.

From now, all the data concerning the number of hidden units and the learning rate is optimal for the learning by the networks.

The network implemented has 10 input units (variations of the 10 integers), 5 hidden units and 2 output units. If the gesture is '*hand opening*', then the inputs are positive (because the integers are growing). The sigmoid function used is $f(x) = (1 + e^{-x})^{-1}$. It takes values between 0 and +1. So there are 2 outputs: g1 and g2. g1 is related to enlargement and g2 to shrinkage. The '*growth*' mentioned above is then g1 - g2. As g1 and g2 are in [0, 1], the *growth* is in [-1, +1]. If *growth* > 0, then it's an enlargement, otherwise it's a shrinkage. When using other sigmoid functions taking values in the range [-1, +1], the network doesn't converge. So we prefered to have two outputs between 0 and +1.

The following figure shows how the data is analysed :

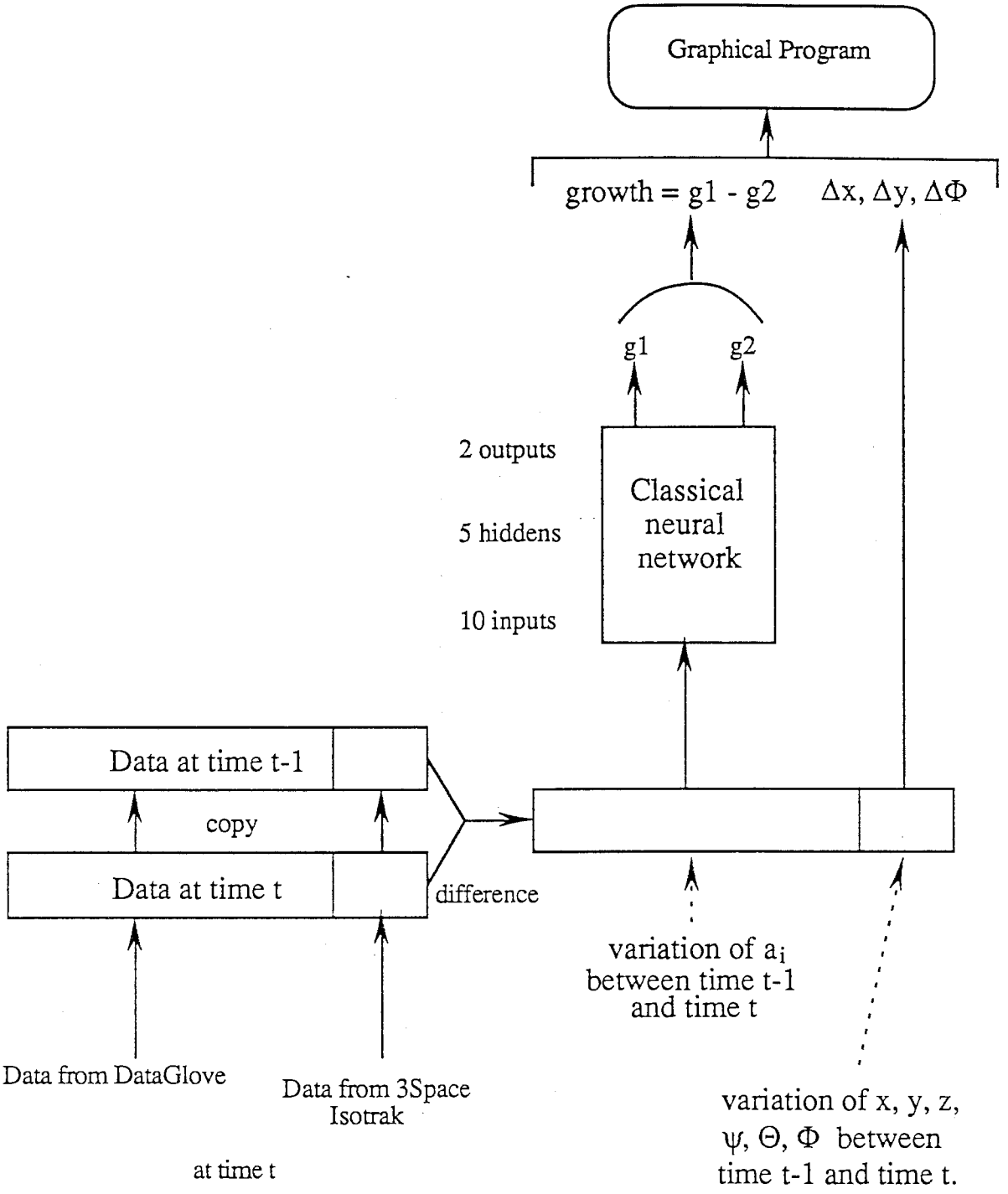
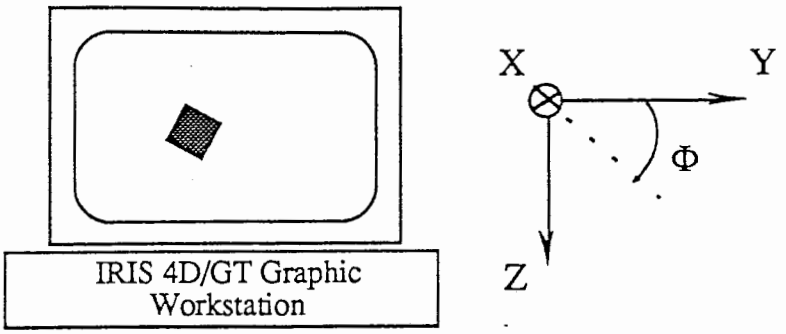


Figure 6 : Method to analyse data.

Remark 1 :

Only x, y and Φ are kept because graphical images in our study are 2 dimensional. The axes are fixed as follows :



So for rotation, Φ is sufficient. (*see fig 3 p. 8 : Euler angles*).

Remark 2 :

Let's call g the *growth*. g is a scaling factor for all lengths. For example, at time t, if the side of a square is $l(t)$, and the growth is g, then at time $t+1$, the side will be :

$$l(t+1) = l(t)*(1 + g).$$

This is the simplest equation. Of course other equatios may be used. A general equation can be :

$$l(t+1) = A*l(t)*h(g) \qquad \text{(eq. 2)}$$

where h is a monotonicaly increasing function,
and A is a scaling factor (positive).

3.3. WHAT ARE THE LIMITATIONS OF THIS METHOD ?

Using a simple neural network, described above, we are able to compute at each time a certain number of parameters to move objects. But in this case, every movement of the hand is interpreted without discontinuity. Now, we would like to stop and restart the interpretation : for instance, if there are several objects and if you are working on an object, you'd like to stop and then after choosing another object, restart analysing hand motion; another example : to enlarge an object, you open your hand. But when it is completely open, you can't enlarge anymore. So we should somehow freeze the network, close the hand, and restart the operation.

Another aspect is *continuous enlargement* (or shrinkage). Experiments have shown that to enlarge continuously an image, subjects tend to do the following gesture : open the hand slowly, close it quickly and repeat this sequence several times. So how to teach the network in order to recognize such complex gesture ? How to represent the past and how to predict next states of the hand ?

These are some issues treated in the following sections.

3.4. NEUTRAL POSITIONS

To stop or restart the interpretation of the hand motion, we defined two particular positions : open hand and closed hand (clenched fist). These positions are called *neutral positions*.

The method is the following : by experiments, data (the 10 integers) for those two positions can be obtained. Values are maximum for open hand, and minimum for closed hand. When getting data from the glove and before presenting them to the input units, they are stored in a table. After one or two seconds, their average values are calculated. If those values are close to maximum or minumum values, then data are not interpreted anymore, because a neutral position has actually been detected. In fact, it's the outputs of the network that are not interpreted. When detecting the next neutral position by the same technique, interpretation is taken up again. (*For more details, see the programs).*

3.5. WORKING ON OUTPUTS

We mentioned in 3.3. the *continuous enlargement* gesture. For this gesture, the evolution of g (*growth*) is like this :

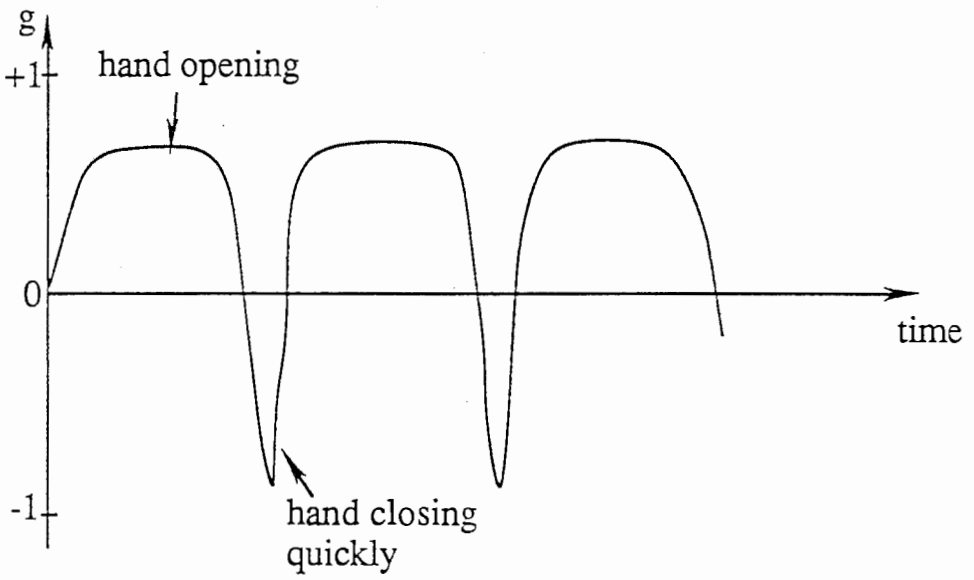


Figure 7 : Evolution of g for continuous enlargement gesture.

Using *eq. 2*, variation of length (side of a square, or radius of a circle, for instance) will have the aspect illustrated in figure 8 :

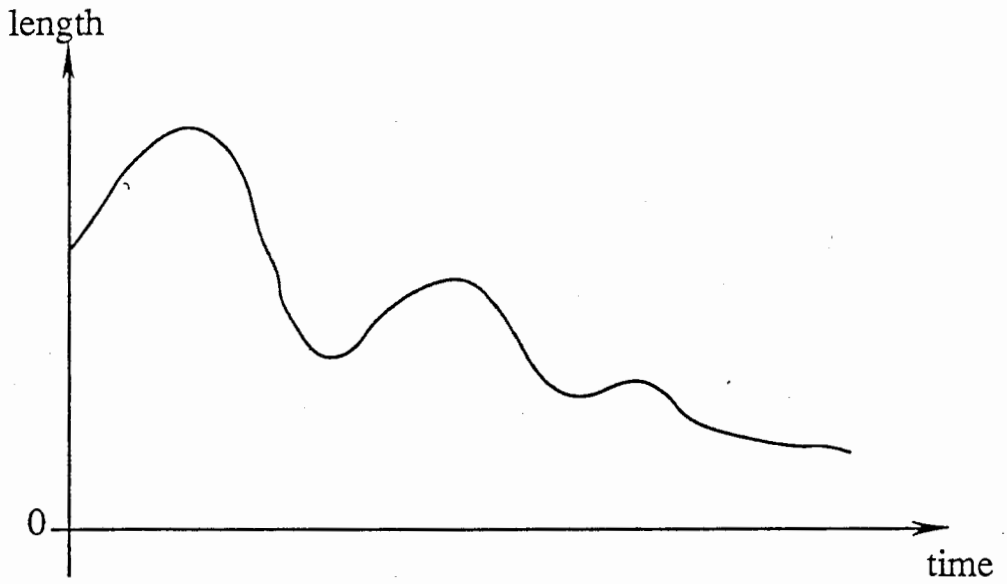


Figure 8 : Evolution of length for continuous enlargement gesture using the simple neural network.

The response, as expected, is not good. One way it to improve is to deal with the output g. Indeed, if we consider the integral of g, and use an equation like :

$$l(t+1) = h(l(t), \int g) \qquad \text{(eq. 3)}$$

where h is a monotonicaly increasing function according to the second variable, better response is obtained, because $\int g$ has this aspect :

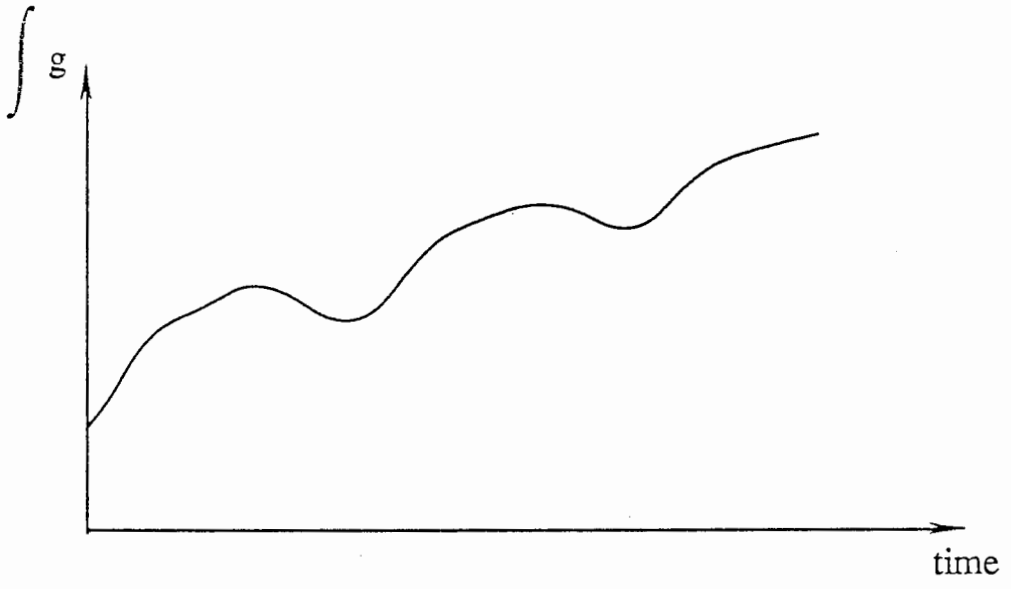
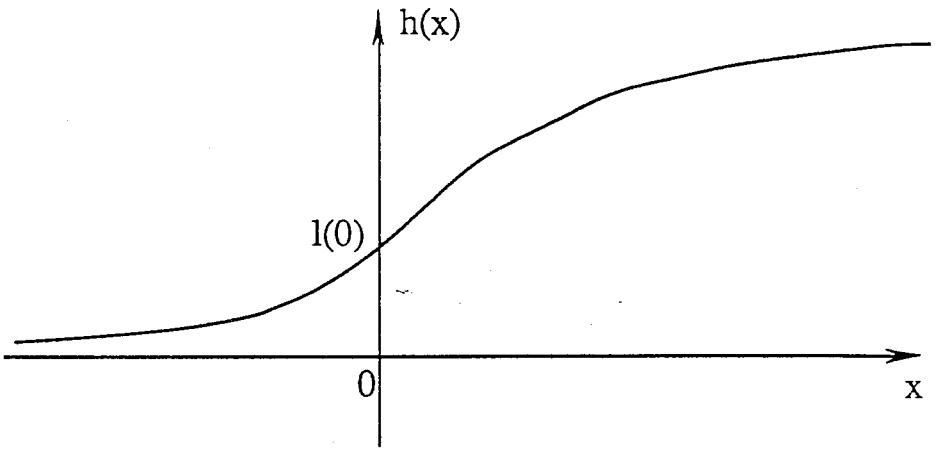


Figure 9 : Variation of $\int g$ for a continuous enlargement gesture.

The function h we used is defined by :

$$\begin{cases} l(t+1) = h(x) = l(0)\times(1 + \frac{2}{\pi}atanx) & \text{if } x < 0 \\ l(t+1) = h(x) = l(0)\times\sqrt{1 + \frac{4}{\pi}x} & \text{if } x > 0 \end{cases}$$

where $x = (\int g)(t)$.



There are other ways to tackle this problem. In fact, the main question is how to represent time in neural networks. The next sections present two different methods.

3.6. USING MORE INPUTS

One way is to represent time *explicitly*. At time t, values of data at time t-1, t-2, ... are also presented to the network. The idea is to take into account the "*past* ".

If many positive inputs are followed by a few negative ones, maybe other positive inputs will come after. It's the case for continuous enlargement gesture (as described previously). So we somehow want to ignore the negative inputs.

Here are some learning patterns :

Inputs value at time				Desired output at time
t-3	t-2	t-1	t	t
+	+	+	+	+++
+	+	+	-	++
+	+	-	+	+
+	+	-	-	0
+	-	-	-	--

+ means positive input (hand opening gesture) or output,
- means negative input (hand closing gesture) or output,
(the number of signs is to show intensity of the value).

One problem is that the number of input nodes is significantly increased. For example, if we input data at times t-2, t-1 and t, the number is 30. So the number of hidden units should be also increased. The number of iterations for learning is then higher, and the learning takes more time.

3.7. SIMPLE RECURRENT NETWORK

The approach described above is to represent time explicitly. Instead of treating time as an explicit part of the input, there is another, very different possibility : to represent time by the effect it has on processing. This means giving the processing system dynamic properties that are responsive to temporal sequences. In short, the network is be given memory. Jordan (*see ref [7]*) described a network containing recurrent connections that are used to associate a static pattern with a serially ordered output (a sequence of "Actions") (*see figure 10*).

The recurrent connections allow the network's hidden units to see its own previous output, so that the subsequent beheviour can be shaped by previous response. These recurrent connections provide the network with memory.

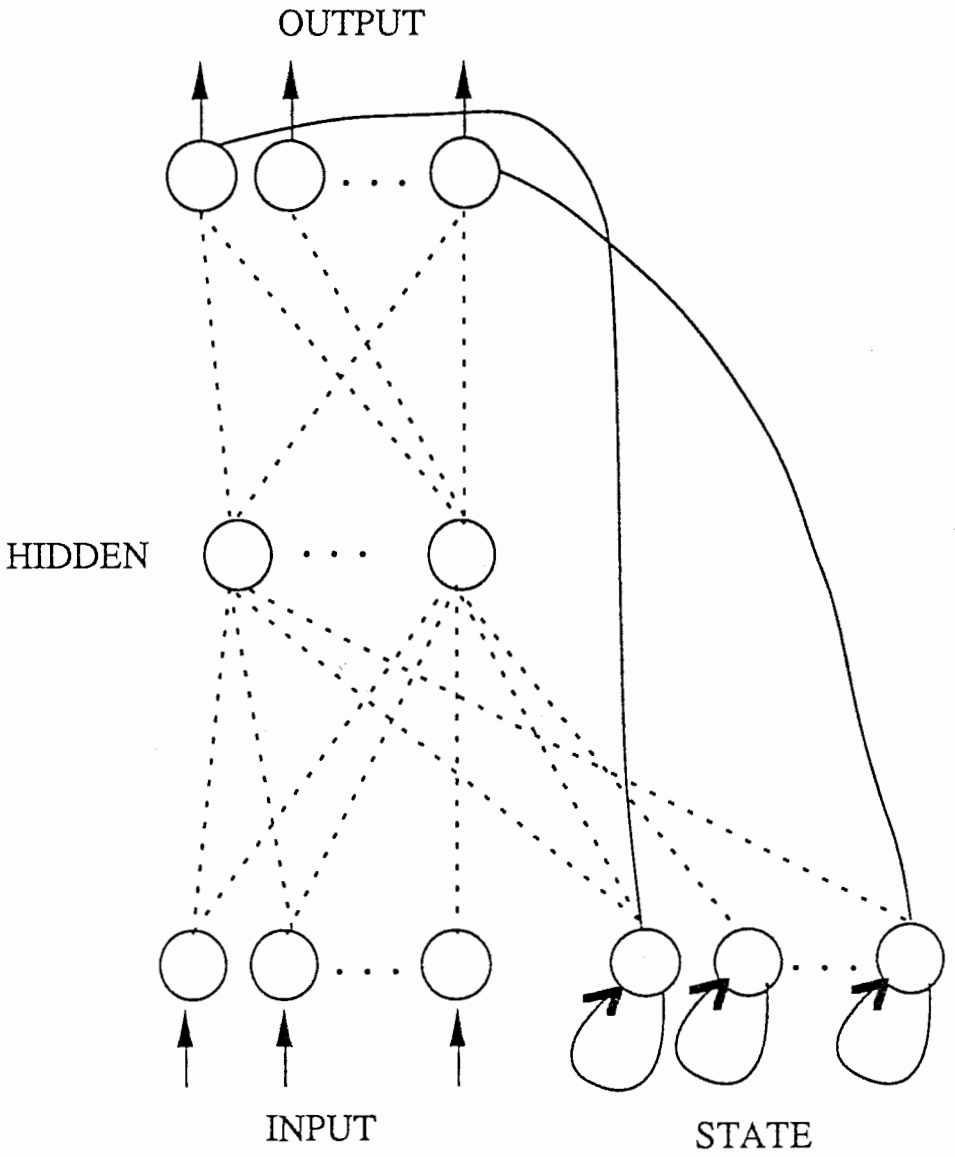


Figure 10 : Architecture used by Jordan (1986).
Connections from output to state units are one-for-one, with a fixed weight of 1.0. Not all the connections are shown.

This approach can be modified in the following way (*proposed by Elman, see ref [8]*). Suppose a network (*shown in figure 11*) is augmented at the input level by additional units; these are called *Context Units*.

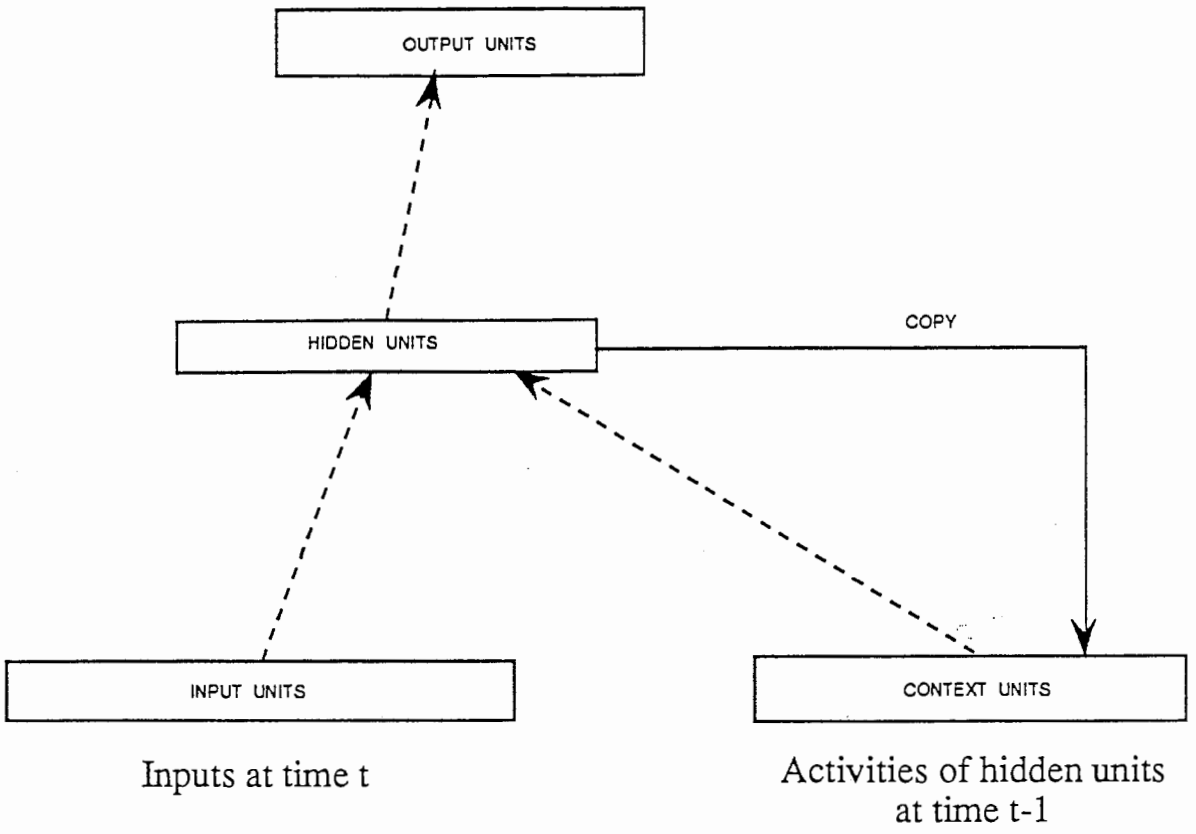


Figure 11 : A simple recurrent network.

These units are also "hidden" in the sense that they interact exclusively with other nodes internal to the network and not the outside world. Activations are copied from hidden layer to context layer on a one-for-one connections, with fixed weight of 1.0. Dotted lines represent trainable connections. The network is taught with indepedent input patterns, but they constitute a whole sequence : the network learns gestures. Here is the method :

At time t = 0, the input units receive the first input of a sequence. The context units are initially set to 0.5 or 0. Both the input units and context units activate the hidden units; the hidden units then feed back to activate the context units (activities of the hidden units are saved in the context units). The output is compared to the desired output and backpropagation of error is used to adjust connection weights (*see algorithm in section 3.2.*). Recurrent connections are fixed at 1.0 and are not subject to adjustement. At the next step, time t + 1, the above sequence is repeated. This time, the context units contain values which are exactly the hidden units values at time t. The context units thus give the network memory.

The network is then able to "predict" evolution of data. For example, if the network is taught the following sequence :

time	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}
input value	+	+	+	+	-	-	+	+	+	+	-	-

(this could be a continuous enlargement gesture), then for a gesture with data which follows a similar sequence, the network may expect a positive input at t_{12}.

Due to errors, one might expect some time delay in the response. But it's negligible compared to the delay there is in the case where we increase the number of inputs (*see experimental results*).

IV. EXPERIMENTAL RESULTS

4.1. DATA FILE

Here is a sample of a data file :
The format is the following :

time a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} x y z Ψ Θ Φ

a_1, a_2 : thumb inner and outer joints
a_3, a_4 : index inner and outer joints
a_5, a_6 : middle inner and outer joints
a_7, a_8 : ring inner and outer joints
a_9, a_{10} : pinky inner and outer joints

x, y, z : coordinates (in centimeter, usually).
Ψ Θ Φ :Euler angles (azimuth, elevation and roll; in degree usually).

This data file is for an arbitrary gesture. We don't fix the frequency of getting data. But we impose the period to be greater than 0.1 second. The networks are trained with patterns according to this frequency. This parameter can of course be modified, depending on what you intend to do.

SAMPLE OF A DATA FILE

	Data from DataGlove (10 integers)										3D Tracker (x, y, z and Euler angles)					
time																
85405.177621	88	83	98	62	71	97	159	62	65	152	-10.61	88.29	23.13	-51.51	35.02	91.33
85405.387621	82	79	97	54	73	80	161	57	67	132	-18.34	87.95	21.21	-22.53	39.13	94.26
85405.547621	80	76	95	54	72	76	161	54	66	129	-27.21	86.80	19.62	-2.42	48.65	95.00
85405.707621	79	75	95	54	72	78	162	54	67	131	-32.64	83.39	19.66	2.61	51.93	90.33
85405.858059	83	90	122	65	77	104	177	68	78	164	-31.37	79.87	14.87	-14.09	41.01	69.19
85406.018059	88	112	150	75	76	124	190	82	84	204	-27.83	76.96	10.83	-19.50	31.33	59.32
85406.168059	108	126	163	80	77	136	195	93	87	223	-23.53	75.54	8.38	-20.65	23.41	55.95
85406.318059	120	132	167	82	79	148	197	96	88	228	-18.61	74.59	6.74	-22.27	13.20	52.00
85406.468059	124	133	168	83	79	152	195	96	88	231	-15.50	74.81	6.24	-20.78	11.13	51.97
85406.618059	103	117	155	80	75	141	185	91	83	220	-16.63	77.49	4.71	-17.91	23.51	57.88
85406.768496	81	75	109	46	76	66	171	49	75	114	-26.34	81.32	10.77	20.76	51.07	99.56
85406.928496	85	88	124	52	79	74	177	53	79	123	-25.13	79.88	15.44	2.23	44.93	85.34
85407.078496	93	114	152	70	79	110	191	73	89	184	-21.25	78.15	12.73	-15.50	31.35	65.45
85407.228496	110	126	166	77	79	129	194	89	90	212	-18.32	77.36	10.98	-18.26	25.48	61.47
85407.388496	111	130	168	80	82	142	197	95	91	222	-14.81	76.59	9.63	-21.63	18.56	59.37
85407.548496	118	131	167	81	82	147	195	96	90	223	-13.47	76.63	8.61	-20.99	15.14	58.51
85407.698496	87	94	127	59	76	92	179	65	77	151	-21.40	80.22	7.44	1.38	40.91	77.97
85407.828933	82	78	122	47	80	65	177	45	80	105	-26.11	81.43	15.70	16.81	49.16	102.36
85407.978933	92	114	150	68	83	108	194	71	90	185	-21.69	79.99	12.80	-14.46	36.15	73.66
85408.108933	111	127	162	78	82	131	194	89	91	213	-18.32	78.37	10.13	-21.31	27.08	64.22
85408.268933	118	132	167	81	84	145	196	95	92	221	-15.36	77.45	7.14	-25.16	18.44	58.78
85408.428933	121	132	166	82	85	149	195	96	91	223	-13.83	77.01	4.78	-24.44	16.35	56.70
85408.578933	112	123	159	80	82	142	187	92	88	218	-17.05	79.37	3.49	-19.80	24.92	59.34
85408.729371	82	75	117	52	80	77	175	55	77	126	-24.93	83.15	11.69	13.33	48.46	95.46
85408.879371	87	95	138	58	85	87	187	61	86	149	-23.74	82.57	14.77	-11.03	46.45	79.63
85409.019371	102	124	161	77	85	126	194	87	90	211	-20.77	80.54	10.32	-22.49	37.10	65.05
85409.159371	113	128	164	79	84	135	194	91	91	216	-17.87	79.56	7.99	-24.86	32.87	60.60
85409.309371	118	130	167	80	84	141	194	93	91	219	-15.32	78.81	5.92	-25.59	25.90	57.18
85409.479371	121	133	168	81	85	146	194	95	90	222	-13.65	78.62	3.42	-26.57	21.81	53.88
85409.619371	87	89	121	55	82	78	174	58	77	135	-20.95	82.29	6.85	-1.43	45.48	80.56
85409.769808	85	79	132	53	84	80	179	54	82	126	-24.05	83.08	15.56	-5.71	46.61	85.86
85409.929808	95	118	153	73	85	118	193	80	89	203	-23.12	81.82	13.43	-19.68	39.45	70.45
85410.069808	109	127	159	78	85	132	194	90	89	214	-20.63	80.83	10.57	-23.91	35.68	65.14
85410.219808	121	132	161	80	86	141	193	93	90	219	-18.02	80.04	8.08	-25.91	31.08	61.84
85410.359808	122	133	165	81	86	147	193	95	90	222	-15.58	79.26	6.28	-26.47	25.56	59.72
85410.519808	123	133	166	82	86	151	193	95	90	223	-13.93	78.55	4.19	-27.03	22.61	57.37
85410.679808	114	126	162	81	84	145	189	93	88	221	-14.95	78.52	2.36	-24.21	28.07	58.08
85410.830246	90	97	128	68	83	112	175	72	78	171	-17.62	79.87	4.22	-15.09	34.94	66.91
85411.000246	86	89	114	57	79	82	172	56	76	125	-20.33	80.69	7.05	-8.48	39.95	75.36
85411.160246	84	86	115	52	80	70	172	49	76	111	-22.13	81.14	8.46	-7.32	40.99	77.74

4.2. PERFORMANCES OF SOME NETWORKS

Our measure of the error for each learning pattern k is

$$E_k = 1/2 \sum_j (t_j - c_j)^2$$

(see eq. 1 page 13).

j = 1,..., number of outputs.

We decided that the network knows a pattern k if

$$E_k \leq 0.003 \qquad \text{(ERROR TOLERANCE = 0.003)}$$

The following table shows some interesting values of some parameters for different networks :

	Network1	Network2	Network3
Learning patterns	20	25	40
Number of input units	10	30	10
Number of hidden units	5	13	7
Number of output units	2	2	2
Learning rate	0.27	0.27	0.22
Number of iterations	4000	4000	6000

Network1 : Simple network

Network2 : Network with inputs at time t-2, t-1 and t. (30 inputs)

Network3 : Recurrent network.

For different gestures, we studied the evolution of length given by those 3 networks. From the outputs g1 and g2, we calculate the growth g= g1 - g2 and then the "length of an object" using :

$$l(t+1) = l(t)\sqrt{1 + g(t)} \qquad \textit{(see eq. 2 p 16)}$$

Remark : if we take $l(t+1) = l(t)\ (1 + g(t))$, length grows too much, so we attenuate the growth by using $\sqrt{1 + g(t)}$. Another way is to change the learning by decreasing output patterns value.

The following figure is for an *arbitrary gesture* :

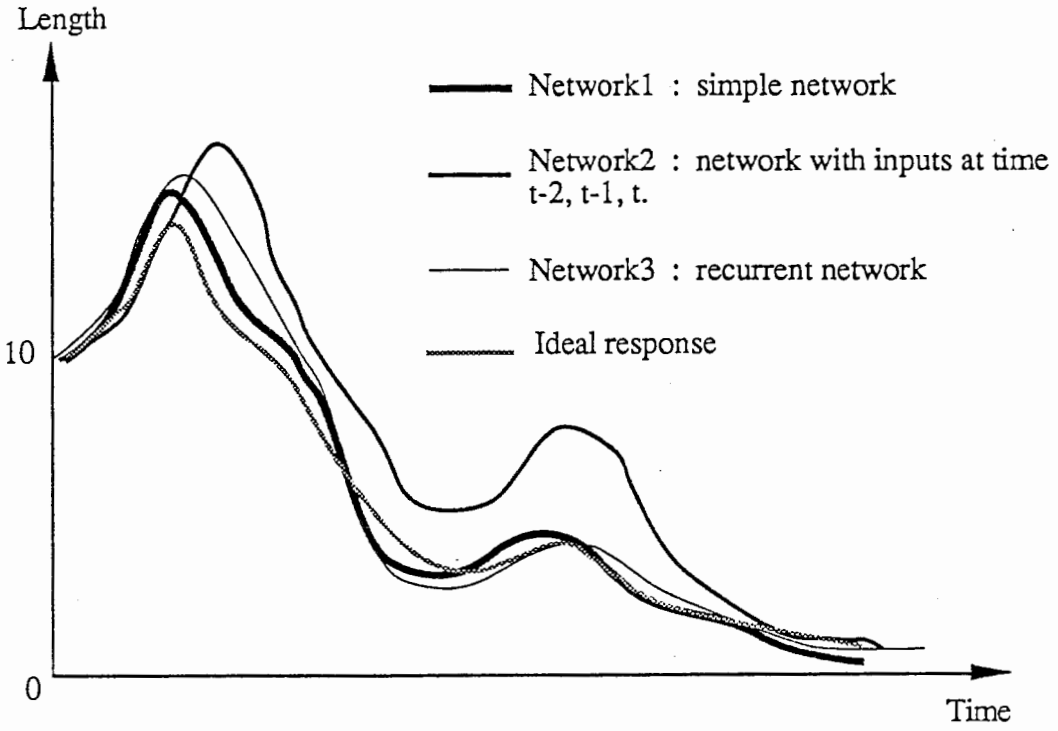


Figure 12 : Evolution of length for an arbitraty gesture.

The ideal response is computed according to what the simple network has been taught. Because this network computes an output for each input, without taking into account previous inputs.

All the curves have a similar shape, but the response given by network 2 is delayed. This is quite logical. This network has been taught in such a way (*see 3.6.*) that this delay is expected.

As we already mentioned, one might expect a delay of this kind for the recurrent network's response. But it's not really the case. That's why we usually say recurrent networks have some power of prediction.

We did the same for a *continuous enlargement gesture* :

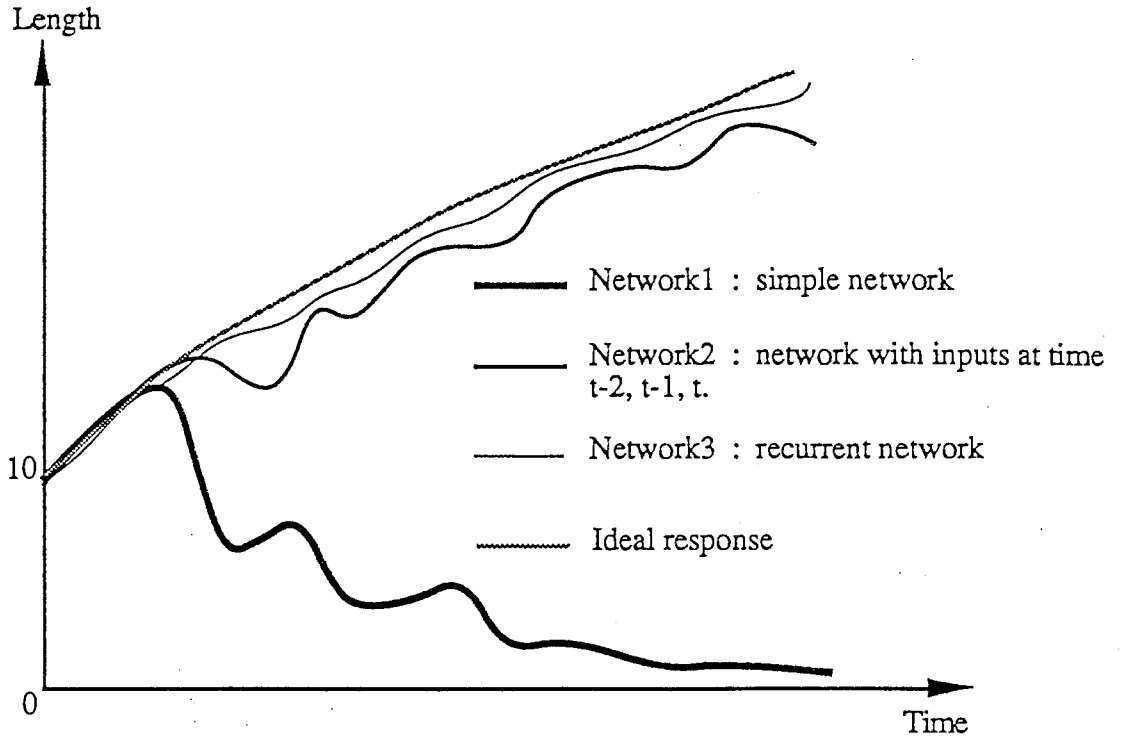


Figure 13 : Evolution of length for continuous enlargement gesture.

The response given by the simple neural network is very bad : when closing the hand quickly, the length decreases too much, so the following opening hand gesture is not enough to enlarge the object. The length drops quickly towards zero.

Responses of network 2 and 3 are better. However, for network 2, oscillations are more stressed than for the recurrent network.

V. GRAPHICAL APPLICATION

According to the previous chapter, the best results are given by the recurrent network. We decided then to use and implement this kind of network. As shown in *figure 6*, outputs of the network are sent to a simple graphical program. We manipulated a filled square and tested if the impression of real-time was satisfactory. The program is run on a IRIS 4D/GT Graphic Workstation.

Obviously, for continuous shrink gestures, responses weren't as good as we expected. We improved the network in two different ways : either by training the network with many more patterns, or by adding another context layer (this network can be called : *Second Order Recurrent Network; see figure 14*).

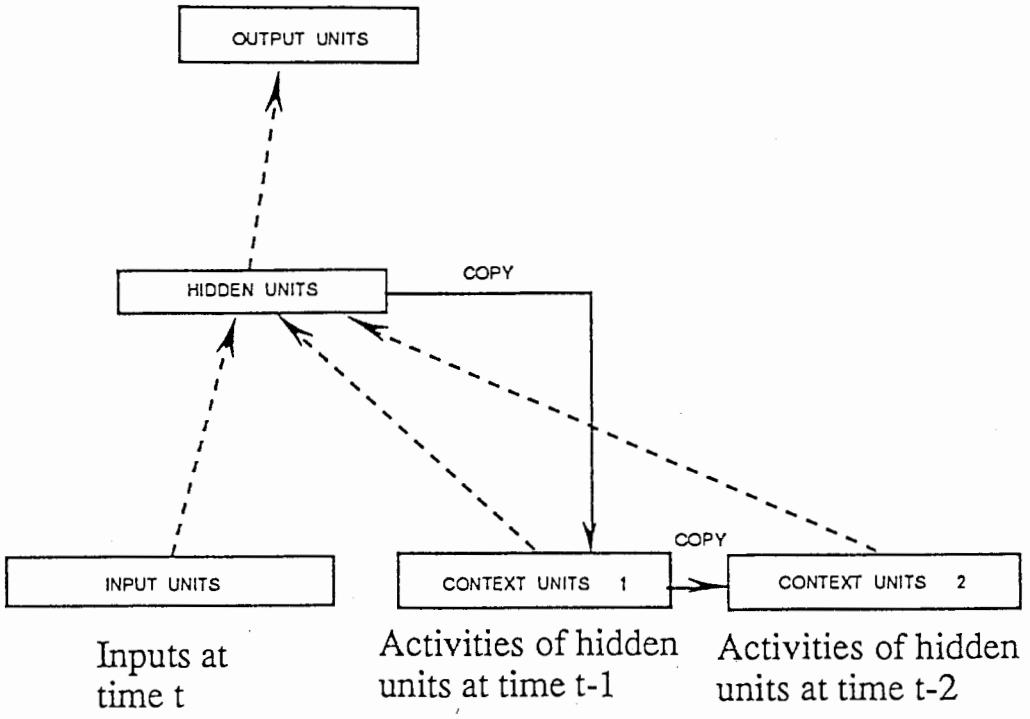


Figure 14 : Second order recurrent network.
The mecanism is the same, except that context units contain hidden units values at time t-1 and t-2.

Here are the performances of these two networks trained with the same set of patterns (*see Appendix 2*) :

	1st Order Recurrent Network	2nd Order Recurrent Network
Learning patterns	70	70
Number of input units	10	10
Number of hidden units	30	23
Learning rate	0.23	0.22
Number of iterations	10000	9000
Training duration	60 s	57 s

Performances are quite similar. But we think that the second order recurrent network has a stronger ability to recognize complex gestures. If we train the network with more patterns, this network might have

better performance than the first order recurrent network. That's why we decided to implement the second order recurrent network. Note that it is possible to go further and implement a third order recurrent network.

V. CONCLUSION

The method described in this report is one way among many to interpret hand gestures in real-time. Because of the difficulties of using data given by the DataGlove, neural networks are used. Evolution of this data is analysed by the network to compute parameters to define the hand motion. To recognize some complex gestures, for instance a continuous enlargement gesture, time should be introduced in the network. It turns out that implicit representation of time in a recurrent network is significantly better than explicit representation. In other words, time, instead of being an explicit part of the inputs, is represented by its effect on processing.

For a complete system, this nonverbal method might be quite limited. Additionally, research on the integration of visual and speech information should be conducted, so that ambiguous instructions can be easily and correctly understood, and the number of operations might be increased.

ACKNOWLEDGEMENTS

This work constitutes the internship report of Gounasségarin MOUROUVAPIN, who would like to express his gratitude to all the people in the Communication Systems Research Laboratories, who have helped him during this internship and contributed to creating a very friendly work atmosphere.

He wishes to express sincere thanks to Takeshi ONISHI for his constant presence and advice during his stay in ATR.

Also he would like to thank people working in the Planning Division for their kindness and constant and effective help.

REFERENCES

[1] DataGlove Model 2 System, Users Manuel, VPL-Research Inc. Redwood City, CA (1987)

[2] 3SPACE Isotrak, Users Manuel, Polhemus, P.O. Box 560, Colchester, VT (1987)

[3] Hauptmann A. G. "Speech and Gestures for Graphic Image Manipulation", CHI '89 Proceedings, Austin Texas pp 241-255 (1989)

[4] Ushiyama H., Matsumoto C., Hiroto K., Murakami K., "Hand Gesture Interpretation using Neural Network", Proceedings of the 40th annual Symposium of IPSJ, pp 152 (1990)

[5] Rumelhart D. E., Mc Clelland J. L., PDP Research Group "Parallel Distributed Processing" volume 1, the MIT Press, Cambridge, Massachusetts (1986)

[6] Simpson P. K., "Artificial Neural Systems", pp 112-116, Pergamon Press (1990)

[7] Jordan M. I. "Attractor Dynamics and Parallelism in a Connectionist Sequential Machine", Proceedings of 8th annual Conference of the Cognitive Science Society. Hillsdale, NJ : Erlbaum (1986)

[8] Elman J. L., "Finding Structure in Time", Cognitive Science 14, pp 179-211 (1990)

APPENDIX 1

THE GENERALIZED DELTA RULE

The learning procedure we propose involves the presentation of a set of pairs of input and output patterns. The system first uses the input vector to produce its own output vector and then compares this with the *desired output*, or *target* vector. If there is no difference, no learning takes place. Otherwise the weights are changed to reduce the difference. In this case, with no hidden units, this generates the standard delta rule . The rule for changing weights following presentation of input/output pair p is given by

$$\Delta_p w_{ji} = \eta (t_{pj} - o_{pj})\, i_{pi} = \eta \delta_{pj} i_{pi} \tag{1}$$

where t_{pj} is the target input for jth component of the output pattern for pattern p, o_{pj} is the jth element of the actual output pattern produced by the presentation of input pattern p, i_{pi} is the value of the ith element of the input pattern, $\delta_{pj} = t_{pj} - o_{pj}$, and $\Delta_p w_{ij}$ is the change to be made to the weight from the ith to the jth unit following presentation of pattern p.

The delta rule and gradient descent. There are many ways of deriving this rule. For present purposes, it is useful to see that for linear units it minimizes the squares of the differences between the actual and the desired output values summed over the output units and all pairs of input/output vectors. One way to show this is to show that the derivative of the error measure with respect to each weight is proportional to the weight change dictated by the delta rule, with negative constant of proportionality. This corresponds to performing steepest descent on a surface in weight space whose height at any point in weight space is equal to the error measure. (Note that some of the following sections are written in italics. These sections constitute informal derivations of the claims made in the surrounding text and can be omitted by the reader who finds such derivations tedious.)

To be more specific, then, let

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \tag{2}$$

be our measure of the error on input/output pattern p and let $E = \sum E_p$ be our overall measure of the error. We wish to show that the delta rule implements a gradient descent in E when the units are linear. We will proceed by simply showing that

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} i_{pi},$$

which is proportional to $\Delta_p w_{ji}$ as prescribed by the delta rule. When there are no hidden units it is straightforward to compute the relevant derivative. For this purpose we use the chain rule to write the derivative as the product of two parts: the derivative of the error with respect to the output of the unit times the derivative of the output with respect to the weight.

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}. \tag{3}$$

The first part tells how the error changes with the output of the jth unit and the second part tells how much changing w_{ji} changes that output. Now, the derivatives are easy to compute. First, from Equation 2

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) = -\delta_{pj}. \tag{4}$$

Not surprisingly, the contribution of unit u_j to the error is simply proportional to δ_{pj}. Moreover, since we have linear units,

$$o_{pj} = \sum_i w_{ji} i_{pi}, \tag{5}$$

from which we conclude that

$$\frac{\partial o_{pj}}{\partial w_{ji}} = i_{pi}.$$

Thus, substituting back into Equation 3, we see that

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} i_{pi} \tag{6}$$

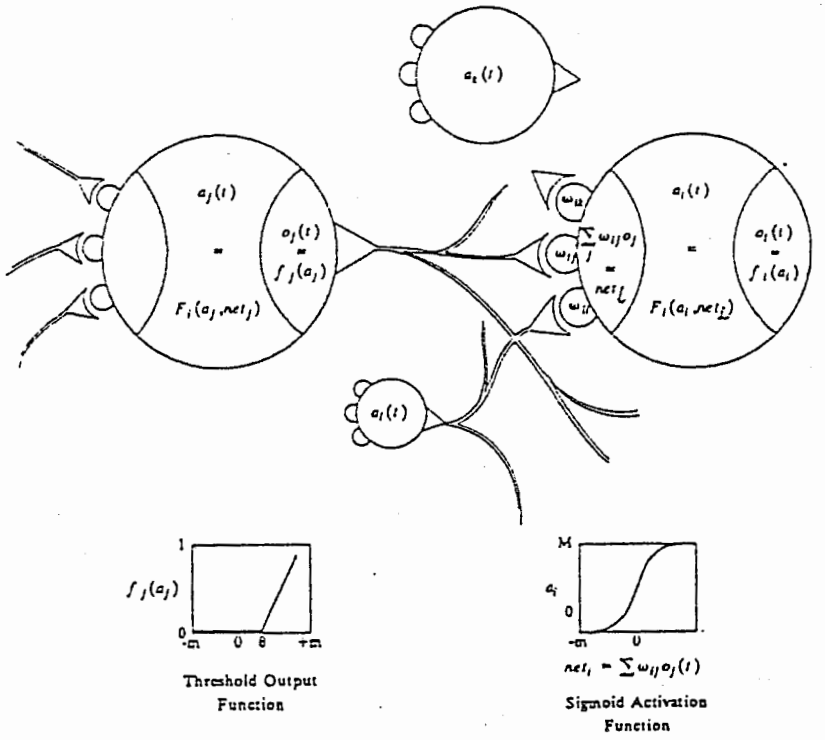
as desired. Now, combining this with the observation that

$$\frac{\partial E}{\partial w_{ji}} = \sum_p \frac{\partial E_p}{\partial w_{ji}}$$

should lead us to conclude that the net change in w_{ji} after one complete cycle of pattern presentations is proportional to this derivative and hence that the delta rule implements a gradient descent in E. In fact, this is strictly true only if the values of the weights are not changed during this cycle. By changing the weights after each pattern is presented we depart to some extent from a true gradient descent in E. Nevertheless, provided the learning rate (i.e., the constant of proportionality) is sufficiently small, this departure will be negligible and the delta rule will implement a very close approximation to gradient descent in sum-squared error. In particular, with small enough learning rate, the delta rule will find a set of weights minimizing this error function.

The delta rule for semilinear activation functions in feedforward networks. We have shown how the standard delta rule essentially implements gradient descent in sum-squared error for linear activation functions. In this case, without hidden units, the error surface is shaped like a bowl with only one minimum, so gradient descent is guaranteed to find the best set of weights. With hidden units, however, it is not so obvious how to compute the derivatives, and the error surface is not concave upwards, so there is the danger of getting stuck in local minima. The main theoretical contribution of this chapter is to show that there is an efficient way of computing the derivatives. The main empirical contribution is to show that the apparently fatal problem of local minima is irrelevant in a wide variety of learning tasks.

At the end of the chapter we show how the generalized delta rule can be applied to arbitrary networks, but, to begin with, we confine ourselves to *layered feedforward* networks. In these networks, the input units are the bottom layer and the output units are the top layer. There can be many layers of hidden units in between, but every unit must send its output to higher layers than its own and must receive its input from lower layers than its own. Given an input vector, the output vector is computed by a forward pass which computes the activity levels of each layer in turn using the already computed activity levels in the earlier layers.



The basic components of a parallel distributed processing system.

Since we are primarily interested in extending this result to the case with hidden units and since, hidden units with linear activation functions provide no advantage, we begin by generalizing our analysis to the set of nonlinear activation functions which we call *semilinear* . A semilinear activation function is one in which the output of a unit is a nondecreasing and differentiable function of the net total output,

$$net_{pj} = \sum_i w_{ji} o_{pi}, \tag{7}$$

where $o_i = i_i$ if unit i is an input unit. Thus, a semilinear activation function is one in which

$$o_{pj} = f_j(net_{pj}) \tag{8}$$

and f is differentiable and nondecreasing. The generalized delta rule works if the network consists of units having semilinear activation functions. Notice that linear threshold units do not satisfy the requirement because their derivative is infinite at the threshold and zero elsewhere.

To get the correct generalization of the delta rule, we must set

$$\Delta_p w_{ji} \propto -\frac{\partial E_p}{\partial w_{ji}},$$

where E is the same sum-squared error function defined earlier. As in the standard delta rule it is again useful to see this derivative as resulting from the product of two parts: one part reflecting the change in error as a function of the change in the net input to the unit and one part representing the effect of changing a particular weight on the net input. Thus we can write

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}}. \tag{9}$$

By Equation 7 we see that the second factor is

$$\frac{\partial net_{pj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_k w_{jk} o_{pk} = o_{pi}. \tag{10}$$

Now let us define

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}}.$$

(By comparing this to Equation 4, note that this is consistent with the definition of δ_{pj} used in the original delta rule for linear units since $o_{pj} = net_{pj}$ when unit u_j is linear.) Equation 9 thus has the equivalent form

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} o_{pi}.$$

This says that to implement gradient descent in E we should make our weight changes according to

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi}, \tag{11}$$

just as in the standard delta rule. The trick is to figure out what δ_{pj} should be for each unit u_j in the network. The interesting result, which we now derive, is that there is a simple recursive computation of these δ's which can be implemented by propagating error signals backward through the network.

To compute $\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}}$, we apply the chain rule to write this partial derivative as the product of two factors, one factor reflecting the change in error as a function of the output of the unit and one reflecting the change in the output as a function of changes in the input. Thus, we have

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}}. \tag{12}$$

Let us compute the second factor. By Equation 8 we see that

$$\frac{\partial o_{pj}}{\partial net_{pj}} = f'_j(net_{pj}),$$

which is simply the derivative of the squashing function f_j for the jth unit, evaluated at the net input net_{pj} to that unit. To compute the first factor, we consider two cases. First, assume that unit u_j is an output unit of the network. In this case, it follows from the definition of E_p that

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}),$$

which is the same result as we obtained with the standard delta rule. Substituting for the two factors in Equation 12, we get

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj}) \tag{13}$$

for any output unit u_j. If u_j is not an output unit we use the chain rule to write

$$\sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial}{\partial o_{pj}} \sum_i w_{ki} o_{pi} = \sum_k \frac{\partial E_p}{\partial net_{pk}} w_{kj} = -\sum_k \delta_{pk} w_{kj}.$$

In this case, substituting for the two factors in Equation 12 yields

$$\delta_{pj} = f'_j(net_{pj}) \sum_k \delta_{pk} w_{kj} \tag{14}$$

whenever u_j is not an output unit. Equations 13 and 14 give a recursive procedure for computing the δ's for all units in the network, which are then used to compute the weight changes in the network according to Equation 11. This procedure constitutes the generalized delta rule for a feedforward network of semilinear units.

These results can be summarized in three equations. First, the generalized delta rule has exactly the same form as the standard delta rule of Equation 1. The weight on each line should be changed by an amount proportional to the product of an error signal, δ, available to the unit receiving input along that line and the output of the unit sending activation along that line. In symbols,

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi}.$$

The other two equations specify the error signal. Essentially, the determination of the error signal is a recursive process which starts with the output units. If a unit is an output unit, its error signal is very similar to the standard delta rule. It is given by

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj})$$

where $f'_j(net_{pj})$ is the derivative of the semilinear activation function which maps the total input to the unit to an output value. Finally, the error signal for hidden units for which there is no specified target is determined recursively in terms of the error signals of the units to which it directly connects and the weights of those connections. That is,

$$\delta_{pj} = f'_j(net_{pj}) \sum_k \delta_{pk} w_{kj}$$

whenever the unit is not an output unit.

The application of the generalized delta rule, thus, involves two phases: During the first phase the input is presented and propagated forward through the network to compute the output value o_{pj} for each unit. This output is then compared with the targets, resulting in an error signal δ_{pj} for each output unit. The second phase involves a backward pass through the network (analogous to the initial forward pass) during which the error signal is passed to each unit in the network and the appropriate weight changes are made. This second, backward pass allows the recursive computation of δ as indicated above. The first step is to compute δ for each of the output units. This is simply the difference between the actual and desired output values times the derivative of the squashing function. We can then compute weight changes for all connections that feed into the final layer. After this is done, then compute δ's for all units in the penultimate layer. This propagates the errors back one layer, and the same process can be repeated for every layer. The backward pass has the same computational complexity as the forward pass, and so it is not unduly expensive.

We have now generated a gradient descent method for finding weights in any feedforward network with semilinear units. Before reporting our results with these networks, it is useful to note some further observations. It is interesting that not all weights need be variable. Any number of weights in the network can be fixed. In this case, error is still propagated as before; the fixed weights are simply not modified. It should also be noted that there is no reason why some output units might not receive inputs from other output units in earlier layers. In this case, those units receive two different kinds of error: that from the direct comparison with the target and that passed through the other output units whose activation it affects. In this case, the correct procedure is to simply add the weight changes dictated by the direct comparison to that propagated back from the other output units.

APPENDIX 2

LEARNING PATTERNS FOR THE SECOND ORDER RECURRENT NETWORK

We teach to the network the following sequence of input/output patterns :

SEQUENCE OF INPUT PATTERNS										TARGET PATTERNS	Comments :
1	3	1	1	1	1	4	1	2	3	0.0	Almost motionless.
0	0	1	0	0	0	0	0	0	1	0.0	
0	0	0	1	0	0	1	0	0	0	0.0	
0	6	2	0	2	2	4	1	3	4	0.0	
-1	3	2	1	0	2	5	1	2	4	0.05	
-1	6	10	5	3	12	16	6	6	15	0.1	Opening hand
0	8	18	17	8	48	18	13	15	21	0.6	
0	9	15	11	2	58	15	15	12	17	0.6	and
-1	2	2	1	1	4	2	5	2	5	0.05	
-6	-20	-17	-28	-2	-103	-12	-30	-10	-53	0.0	Closing hand quickly
-4	-14	-24	-7	-11	-19	-25	-11	-13	-18	0.0	
4	13	18	10	7	23	21	12	9	29	0.35	This is a conti-
2	9	11	14	5	51	8	14	7	22	0.28	nuous enlargement
1	4	6	8	1	34	5	5	4	8	0.6	gesture.
1	10	10	6	0	19	8	12	7	13	0.4	
-5	-24	-26	-32	-3	-114	-20	-35	-15	-56	0.0	
-2	-12	-14	-5	-7	-10	-14	-7	-7	-12	0.0	
4	18	19	14	7	36	20	17	9	38	0.3	
2	7	11	13	5	53	8	14	8	17	0.3	
0	9	13	10	0	33	9	15	8	17	0.75	
3	3	-2	2	1	2	-6	1	-2	3	0.0	
-9	-31	-38	-39	-11	-126	-24	-47	-21	-75	0.0	
0	0	0	0	-1	1	-3	2	0	5	0.0	
5	16	20	18	9	45	18	19	10	38	0.35	
1	6	11	12	2	51	8	10	8	11	0.2	
1	9	8	6	0	21	6	10	5	14	0.4	End of that gesture.
2	5	2	1	0	4	0	5	1	6	0.3	
0	0	-1	1	0	0	0	0	0	1	0.0	Fixed position.
0	0	0	0	0	0	0	0	0	0	0.0	
0	0	4	0	2	1	9	1	5	2	0.0	
6	25	31	9	10	22	25	18	19	30	0.3	Enlargement.
13	6	7	5	4	9	10	8	-2	5	0.5	
-16	-22	-6	-4	-3	-9	-6	-3	-6	-24	0.0	
-4	-8	-27	-9	-8	-21	-12	-17	-14	-24	0.0	
-5	-5	-10	-23	-3	-52	-3	-27	-5	-45	-0.85	It's effectively
-1	-1	-5	-3	-2	-6	-5	-3	-1	-6	-0.7	a shrink.
9	21	19	27	6	59	16	38	14	56	0.0	
14	17	9	7	8	23	12	16	8	25	0.0	It's effectively
16	17	9	7	3	31	10	9	6	15	0.85	an enlargement.
-2	-5	-2	-1	0	-4	0	-5	-1	-6	0.0	
-1	-9	-8	-6	0	-18	-6	-10	-5	-14	0.0	The following
-1	-6	-11	-12	-2	-54	-8	-10	-8	-11	-0.8	sequence is a
-5	-16	-20	-18	-9	-45	-18	-19	-10	-38	-0.85	continuous shrink
0	0	0	0	1	-2	-3	-2	0	-5	0.0	gesture.
9	31	38	39	11	127	24	47	21	75	0.0	
-3	-3	2	-2	-1	-2	-2	-1	2	-3	-0.4	
0	-9	-13	-10	0	-33	-9	-15	-8	-17	-0.4	
-2	-7	-11	-13	-5	-53	-8	-14	-8	-17	-0.6	
-4	-18	-19	-14	-7	-36	-20	-17	-9	-38	-0.85	
2	12	14	5	7	10	14	7	7	12	0.0	Opening hand
5	24	26	32	3	114	20	35	15	56	0.0	quickly.
-1	-10	-10	-6	0	-19	-8	-12	-7	-13	-0.5	
-1	-4	-6	-8	-1	-34	-5	-5	-4	-8	-0.3	
-2	-9	-11	-14	-5	-51	-8	-14	-7	-22	-0.6	
-4	-13	-18	-10	-7	-23	-21	-12	-9	-29	-0.75	
4	14	24	7	11	19	25	11	13	18	0.0	
6	20	17	28	2	103	12	30	10	53	0.0	
1	-2	-2	-1	-1	-4	-2	-5	-2	-5	-0.3	
0	-9	-15	-11	-2	-58	-15	-15	-12	-17	-0.4	
0	-8	-18	-17	-8	-48	-18	-13	-15	-21	-0.85	End of that gesture.
0	0	0	0	0	0	0	0	0	0	0.0	
0	0	0	0	0	0	0	0	0	0	0.0	Fixed position.
0	0	0	0	0	0	0	0	0	0	0.0	
4	16	18	13	7	34	19	16	9	35	0.3	Enlargement.
13	17	38	39	11	98	25	50	21	70	0.75	
6	20	17	25	5	70	12	30	15	48	0.5	
-5	-25	-24	-30	-7	-80	-25	-40	-20	-50	0.0	
-4	-20	-20	-18	-9	-48	-20	-20	-15	-39	0.0	
-4	-13	-18	-9	-6	-20	-20	-15	-12	-22	-0.75	It's effectively
-4	-9	-11	-8	-5	-10	-12	-13	-12	-17	-0.82	a shrink.

COMPUTER PROGRAMS

SOME DETAILS

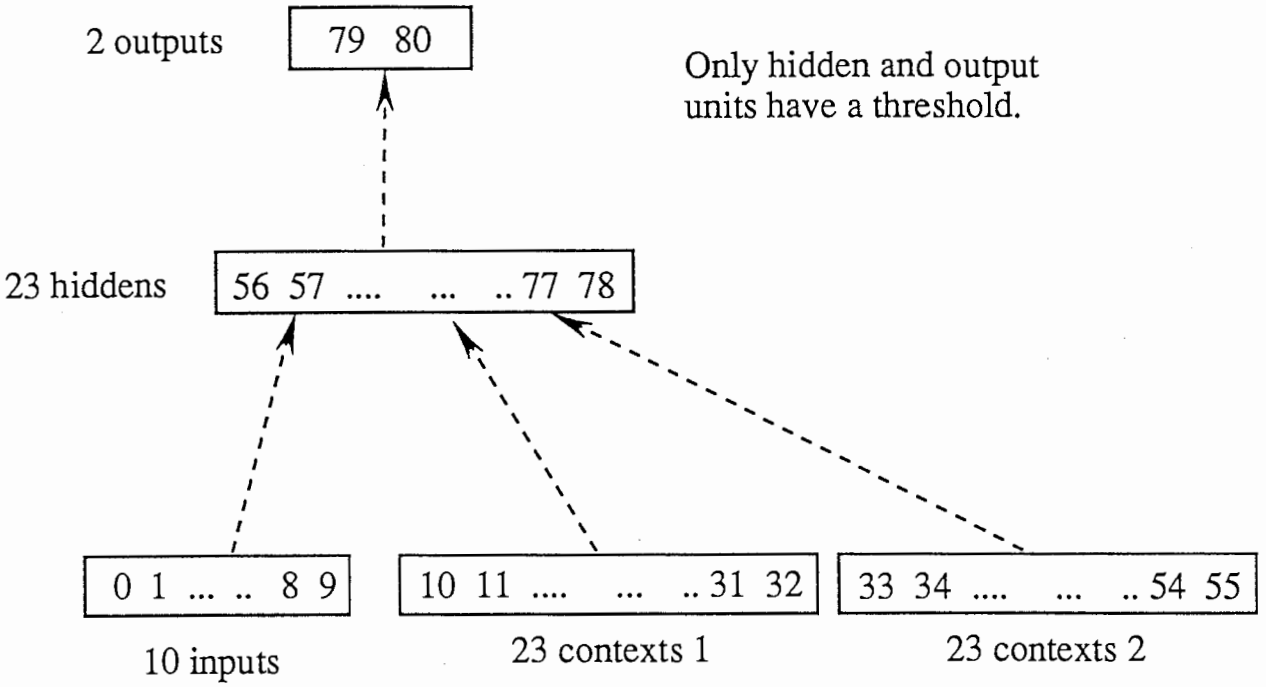
* Programs are run on a IRIS 4D/GT Graphic Workstation (MIPS C Compiler).

* Programs concerning the DataGlove are written for general use, so many macros are defined. But we use only *FLEX_ON* mode, and *USE_SYSTEM_V* (System V compiler).

* The DataGlove is calibrated using the program *dg2_calibrate*. The calibration is user dependent. This program creates a file called BRIGHT_FILE which determines the brightness level in the fibers. In that case, the learning has to be user dependent as well. To avoid it, we trained the network with the default bright level (which is 0). With the latter value, variations of a_i are sometimes quite high (around ± 100), then with the sigmoid function $f(x) = (1 + e^{-x})^{-1}$, activities of hidden and output units are close to ± 1. Learning is impossible. So we defined a scaling factor *SCALE == 0.1* to reduce the range of variations around ± 10. So the inputs of the network are not exactly $(\Delta a_i)_{i = 1,..,10}$, but with a factor *SCALE*. If the DataGlove is not calibrated with the default value, then *SCALE* should be changed to have (Δa_i) around ± 10.

* In the program *network.c*, we defined a function called *save_weights*, that saves all the weights and thresholds into a file WEIGHTS_FILE. Thus the network doesn't need to be trained every time. Weights and thresholds are loaded directly in the network from that file. The order in which weights are saved is as follows :

Suppose the network is :



The order is :

threshold of 56
weight of (56, 0)
.
.
weight of (56, 55)
threshold of 57
weight of (57, 0)
.
.
weight of (57, 55)
.
.
.
threshold of 79
weight of (79, 56)
.
weight of (79, 78)
threshold of 80
weight of (80, 56)
.
.
weight of (80, 78)

The WEIGHTS_FILE has 1359
lines = 10*23 + (23*23)*2 + 23*2 + 23 + 2

weights

thresholds

```
/************************************ MAIN PROGRAM ************************************
***************************************** hand.c *****************************************
*****                                                                                *****
*****              This is the main program : you can train the network or           *****
*****         load it with weights and thresholds from the file "WEIGHTS_FILE".      *****
*****         Data from DataGlove and 3Space Isotrak are sent to the network and     *****
*****         the outputs are used to move a square on the screen.                   *****
*****                                                                                *****
**********************************************************************************************
**********************************************************************************************/

#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <device.h>
#include <gl.h>
#include <math.h>
#include "dg2cmd.h"
#include "dg2def.h"
#include "dg2ext.h"
#include "dg2str.h"
#include "dg2use.h"
#include "tracker.h"
#include "get_time.h"
#include <sys/types.h>
#include <fcntl.h>
#include "network.h"

#define KEY 1965

  char name[40];
  FILE *fp;
  struct unit *pu[TOTAL];
  double tampon1[NUM_CON1], tampon2[NUM_CON2];
  extern double input_pat[PATTERNS+1][NUM_IN];

main()
{
  char ch;
  extern struct unit *pu[];
  extern double tampon1[], tampon2[];

  print_header();

  ch = get_command("¥n¥tDo you want to train a new recurrent network (t) ¥n¥tor load weights to the
network from the file WEIGHTS_FILE (l)¥n¥n");
  if (ch == 't' || ch == 'T') {
     create_processing_units(pu, tampon1, tampon2);
     create_in_out_links(pu);
     for (;;) {
        ch = get_command("¥nEnter Command (Learn(l), Save Weights(s), Recognize(r), Quit(q)) => ");
      switch (ch) {
      case 'l':
      case 'L':
        printf("¥n¥tLEARN MODE¥n¥n");
        learn(pu, tampon1, tampon2);
        break;
      case 's':
      case 'S':
        printf("¥n¥tSaving the weights into the file WEIGHTS_FILE¥n¥n");
        save_weights(pu);
        printf("OK¥n");
        break;
      case 'r':
      case 'R':
        printf("¥n¥tRECOGNIZE MODE¥n¥n");
        movement(pu, tampon1, tampon2);
        break;
      case 'q':
      case 'Q':
        exit(1);
        break;
      default:
        fprintf(stderr, "Invalid Command¥n");
        break;
      }
     }

  } else if (ch == 'l' || ch == 'L') {

     fflush(stderr);
     printf("¥n¥tLoading weights to the network from the file WEIGHTS_FILE¥n¥n");
     create_processing_units(pu, tampon1, tampon2);
     load_weights(pu);
     for (;;) {
        ch = get_command("¥nEnter Command (Learn(l), Save Weights(s), Recognize(r), Quit(q)) => ");
      switch (ch) {
      case 'l':
      case 'L':
        printf("¥n¥tLEARN MODE¥n¥n");
        learn(pu, tampon1, tampon2);
        break;
      case 's':
      case 'S':
        printf("¥n¥tSaving the weights into the file WEIGHTS_FILE¥n¥n");
        save_weights(pu);
        printf("OK¥n");
        break;
```

```
      case 'r':
      case 'R':
        printf("¥n¥tRECOGNIZE MODE¥n¥n");
        movement(pu, tampon1, tampon2);
        break;
      case 'q':
      case 'Q':
        exit(1);
        break;
      default:
        fprintf(stderr, "Invalid Command¥n");
        break;
      }
    }

  } else {

    fprintf(stderr, "Invalid Command¥n");
    exit(1);
  }
}


movement(pu, tampon1, tampon2)
struct unit *pu[];
double tampon1[], tampon2[];
{
  int i, j, idx, m, n, loop, read_length, sauve = 0;
  float y0 = XMAXSCREEN/4, z0 = YMAXSCREEN/4, s0 = 100.0, r0 = 0.0;
                /* y0 and z0 : center of the window; s0 : side of the square or
                   radius of the cercle and r0 : inclination of the square */

  float dy, dz, dr;             /* to stock the variation of x, y and ang */
  double x, y, z, a, e, r, t1, t2, period = 0.1;
  static int index = 0, mode = 1;
                /* mode == 1 => network ON ; mode == 0 => network OFF */
  static int begin = 1;
  static        Hall            hl;
  static        Polhemus        ph;
  static        FlexInnerBright fib;
  static        FlexOuterBright fob;
  static        FlexInnerDim    fid;
  static        FlexOuterDim    fod;

  double input1[NUM_IN + 3], input2[NUM_IN + 3], aver_output[NUM_IN], table[LENGTH][NUM_IN];
  char ch, name[40];
  FILE *fp;
  short val;
  long dev;
  char *buf, tbuf[1024];
  int pid, shmid;
  char *shmat(), *head, *outdata, *tmpbuf = "EXIT";

    ch = get_command("¥nDo you want to save the data into a file ?¥n");
    if (ch == 'y' || ch == 'Y') {
       printf("Name of the file :¥n");
       fflush(stderr);
       gets(name);
       fp = fopen(name,"w");
       sauve = 1;
    }
    loop = double_init();

    printf("¥nOK ! Press RETURN key to start.¥n");
    getchar();
    fflush(stderr);

if (fork() != 0) {
  if ((shmid = shmget(KEY, 1024, 0666 | IPC_CREAT)) == EOF) {
     perror("shmget");
     exit();
  }

  if ((head = outdata = shmat(shmid, 0, 0)) == EOF) {
    perror("shmat");
    exit();
  }

  bzero(outdata, 1024);
  if (shmdt(head) == EOF) {
    perror("shmdt");
    exit();
  }

  while(1) {
    if ((head = outdata = shmat(shmid, 0, 0)) == EOF) {
      perror("shmat");
      exit();
    }
    bcopy(outdata, tbuf, 1024);
    if (shmdt(head) == EOF) {
      perror("shmdt");
      exit();
    }

    if (!strcmp(tbuf, tmpbuf))
      break;
  }
  return(0);
```

```
}
else {
  init_screen();          /* see dessin.c */
  cpack(0);
  clear();
  cpack(0x00ff00);
  draw_square(y0, z0, s0, r0);
/*  draw_cercle(y0, z0, s0);*/
  swapbuffers();

  if ((shmid = shmget(KEY, 1024, 0666 | IPC_CREAT)) == EOF) {
     perror("shmget");
     exit();
  }

  for (i = 0; i < NUM_IN; i++)
      aver_output[i] = 0.0;

  t1 = get_double_time();

  for (i = 1; i < 84; i++)
      qdevice(i);

  for (idx = 0; ; idx++) {

    do {
       t2 = get_double_time();
    } while(t2-t1 <= period);

    if (index == LENGTH)
        index = 0;

/**  Data Get from DataGlove2 & 3SpaceTracker  **/
    read_length = Get_OneShot(&hl, &ph, &fib, &fob, &fid, &fod);
                                  /* because CONTROL == 1 */
    trposin();
    for (m = 0; m < loop; m++) {
       n = trposrotrd(&x, &y, &z, &a, &e, &r);
                           /* 3 coordinates and 3 angles (azimuth, elevation, roll) */
                           /* trposrotrd is defined in tracker.c                    */
    }
    if (sauve) {
       fprintf(fp, "%f ", get_double_time());
    for (i = 0; i < FINGER_SIZE; i++) {
       fprintf(fp, "%3d %3d ",
                    fib.finger[i], fob.finger[i]);
    }
    fprintf(fp, "%7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf¥n",
                  x, y, z, a, e, r);
    fflush(fp);
    }
    for (i = 0, j = 0; i < NUM_IN; i = i+2, j++) {
        input2[i] = fib.finger[j];
        input2[i+1] = fob.finger[j];
    }
    input2[NUM_IN] = y;
    input2[NUM_IN + 1] = z;
    input2[NUM_IN + 2] = r;

    if (begin) {
        for (i = 0; i < NUM_IN + 3; i++)
            input1[i] = input2[i];
        begin = 0;
    }
    for (i = 0; i < NUM_IN; i++) {
        table[index][i] = input2[i];
        input_pat[PATTERNS][i] = SCALE * (input2[i] - input1[i]);
    }
    dy = input2[NUM_IN] - input1[NUM_IN];
    dz = input2[NUM_IN + 1] - input1[NUM_IN + 1];
    dr = input2[NUM_IN + 2] - input1[NUM_IN + 2];

    if (dr >= 300.0)
        dr = dr - 360.0;                /* because e is in [-180.0; 180.0] */
    if (dr <= -300.0)
        dr = dr + 360.0;

    for (i = 0; i < NUM_IN + 3; i++)
        input1[i] = input2[i];

    if (index >= SAMPLE) {
       for (i = 0; i < NUM_IN; i++) {
          for (j= -SAMPLE; j < 0; j++)
             aver_output[i] += table[index + j][i]/SAMPLE;
       }
       if ((freeze(aver_output)) == 1) {
          index = -1;
          if (mode == 1)
              mode = 0;
          else
              mode = 1;
       }
       for (i = 0; i < NUM_IN; i++)
           aver_output[i] = 0.0;
    }
    index++;

    t1 = get_double_time();
```

```
    recognize(pu, tampon1, tampon2, mode, dy, -dz, -dr); /* due to the orientation of the    */
                                                         /* reference related to the Tracker */
    if (qtest()) {
        dev = qread(&val);
        if (dev > 0 && dev < 85)
                break;
    }
  }
    if (sauve)
        fclose(fp);

  if ((shmid = shmget(KEY, 1024, 0666 | IPC_CREAT)) == EOF) {
       perror("shmget");
       exit();
  }

  if ((head = outdata = shmat(shmid, 0, 0)) == EOF) {
       perror("shmat");
       exit();
  }

  buf = "EXIT";
  bcopy(buf, outdata, 4);
  if (shmdt(head) == EOF) {
    perror("shmdt");
    exit();
  }
  exit();
}
}

recognize(pu, tampon1, tampon2, mode, dy, dz, dang)
struct unit *pu[];
double tampon1[], tampon2[];
int mode;
float dy, dz, dang;
{
  static float y = XMAXSCREEN/4, z = YMAXSCREEN/4, s = 100.0, ang = 0.0;
  double growth;

  init_input_units(pu, PATTERNS, tampon1, tampon2);
  propagate(pu, tampon1, tampon2);

  if (mode == 1) {
     growth = pu[OUT_UID(0)]->output - pu[OUT_UID(1)]->output;
     if (growth < 0.01 && growth > -0.01)
         growth = 0.0;
     s *= sqrt(1 + growth);
     y += dy*5;
     z += dz*5;
     ang += dang;
     cpack(0);
     clear();
     cpack(0x00ff00);
   /*  draw_cercle(y, z, s);*/
     draw_square(y, z, s, ang);
     swapbuffers();
  }
}
```

```
/*****************************************************************************
************************************ network.h ******************************
*****                                                                       *****
*****                          Network Library Header File                  *****
*****                                                                       *****
*****************************************************************************
*****************************************************************************/

#define BUFSIZE         512

#define NUM_IN          10      /* number of input units */
#define NUM_CON1        23      /* number of context units for time t-1 */
#define NUM_CON2        23      /* number of context units for time t-2 */
#define NUM_HID         23      /* number of hidden units */
#define NUM_OUT         2       /* number of output units */
#define TOTAL           (NUM_IN + NUM_CON1 + NUM_CON2 + NUM_HID + NUM_OUT)

/***********************************************
  macros to provide indexes for processing units
***********************************************/
#define IN_UID(X)       (X)
#define CON_UID1(X)     (NUM_IN + X)
#define CON_UID2(X)     (NUM_IN + NUM_CON1 + X)
#define HID_UID(X)      (NUM_IN + NUM_CON1 +NUM_CON2 + X)
#define OUT_UID(X)      (NUM_IN + NUM_CON1 + NUM_CON2 + NUM_HID + X)
#define TARGET_INDEX(X) (X - (NUM_IN + NUM_CON1 + NUM_CON2 + NUM_HID))

#define PATTERNS        70      /* number of input patterns */
#define ERROR_TOLERANCE 0.003
#define NOTIFY          10      /* iterations per dot notification */
#define DEFAULT_ITER    8000
#define SAMPLE          8
#define LENGTH          1000
#define SCALE           0.1

struct unit {                   /* general processing unit */
  int    uid;                   /* each unit is identified by an integer */
  char   *label;
  double output;                /* activation level */
  double (*unit_out_f)();       /* activation function*/
  double delta;                 /* delta for unit */
  double (*unit_delta_f)();     /* function to calculate delta */
  double threshold;             /* threshold of the unit */
  struct link *inlinks;         /* for propagation */
  struct link *outlinks;        /* for back propagation */
};

struct link {                   /* link between two processing units */
  char   *label;
  double weight;                /* link weight */
  double data;                  /* used to hold the change in weights */
  int    from_unit;             /* uid of from unit */
  int    to_unit;               /* uid of to unit */
  struct link *next_inlink;
  struct link *next_outlink;
};

#define LEARNING_RATE     0.22
#define MOMENTUM          0.9
```

```
/*****************************************************************************************
****************************************** network.c *************************************
*****                                                                                   *****
*****      Here are defined some functions concerning the Recurrent Network.            *****
*****                                                                                   *****
*****************************************************************************************
*****************************************************************************************/

#include "network.h"

#include <math.h>
#include <stdio.h>
#include <ctype.h>

/* Input Patterns */
double  input_pat[PATTERNS+1][NUM_IN] = {
  {0.1, 0.3, 0.1, 0.1, 0.1, 0.1, 0.4, 0.1, 0.2, 0.3},
  {0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1},
  {0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0},
  {0.0, 0.6, 0.2, 0.0, 0.2, 0.2, 0.4, 0.1, 0.3, 0.4},
  {-0.1, 0.3, 0.2, 0.1, 0.0, 0.2, 0.5, 0.1, 0.2, 0.4},
  {-0.1, 0.6, 1.0, 0.5, 0.3, 1.2, 1.6, 0.6, 0.6, 1.5},
  {0.0, 0.8, 1.8, 1.7, 0.8, 4.8, 1.8, 1.3, 1.5, 2.1},
  {0.0, 0.9, 1.5, 1.1, 0.2, 5.8, 1.5, 1.5, 1.2, 1.7},
  {-0.1, 0.2, 0.2, 0.1, 0.1, 0.4, 0.2, 0.5, 0.2, 0.5},
  {-0.6, -2.0, -1.7, -2.8, -0.2, -10.3, -1.2, -3.0, -1.0, -5.3},
  {-0.4, -1.4, -2.4, -0.7, -1.1, -1.9, -2.5, -1.1, -1.3, -1.8},
  {0.4, 1.3, 1.8, 1.0, 0.7, 2.3, 2.1, 1.2, 0.9, 2.9},
  {0.2, 0.9, 1.1, 1.4, 0.5, 5.1, 0.8, 1.4, 0.7, 2.2},
  {0.1, 0.4, 0.6, 0.8, 0.1, 3.4, 0.5, 0.5, 0.4, 0.8},
  {0.1, 1.0, 1.0, 0.6, 0.0, 1.9, 0.8, 1.2, 0.7, 1.3},
  {-0.5, -2.4, -2.6, -3.2, -0.3, -11.4, -2.0, -3.5, -1.5, -5.6},
  {-0.2, -1.2, -1.4, -0.5, -0.7, -1.0, -1.4, -0.7, -0.7, -1.2},
  {0.4, 1.8, 1.9, 1.4, 0.7, 3.6, 2.0, 1.7, 0.9, 3.8},
  {0.2, 0.7, 1.1, 1.3, 0.5, 5.3, 0.8, 1.4, 0.8, 1.7},
  {0.0, 0.9, 1.3, 1.0, 0.0, 3.3, 0.9, 1.5, 0.8, 1.7},
  {0.3, 0.3, -0.2, 0.2, 0.1, 0.2, -0.6, 0.1, -0.2, 0.3},
  {-0.9, -3.1, -3.8, -3.9, -1.1, -12.6, -2.4, -4.7, -2.1, -7.5},
  {0.0, 0.0, 0.0, 0.0, -0.1, 0.1, -0.3, 0.2, 0.0, 0.5},
  {0.5, 1.6, 2.0, 1.8, 0.9, 4.5, 1.8, 1.9, 1.0, 3.8},
  {0.1, 0.6, 1.1, 1.2, 0.2, 5.1, 0.8, 1.0, 0.8, 1.1},
  {0.1, 0.9, 0.8, 0.6, 0.0, 2.1, 0.6, 1.0, 0.5, 1.4},
  {0.2, 0.5, 0.2, 0.1, 0.0, 0.4, 0.0, 0.5, 0.1, 0.6},
  {0.0, 0.0, -0.1, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
  {0.0, 0.0, 0.4, 0.0, 0.2, 0.1, 0.9, 0.1, 0.5, 0.2},
  {0.6, 2.5, 3.1, 0.9, 1.0, 2.2, 2.5, 1.8, 1.9, 3.0},
  {1.3, 0.6, 0.7, 0.5, 0.4, 0.9, 1.0, 0.8, -0.2, 0.5},
  {-1.6, -2.2, -0.6, -0.4, -0.3, -0.9, -0.6, -0.3, -0.6, -2.4},
  {-0.4, -0.8, -2.7, -0.9, -0.8, -2.1, -1.2, -1.7, -1.4, -2.4},
  {-0.5, -0.5, -1.0, -2.3, -0.3, -5.2, -0.3, -2.7, -0.5, -4.5},
  {-0.1, -0.1, -0.5, -0.3, -0.2, -0.6, -0.5, -0.3, -0.1, -0.6},
  {0.9, 2.1, 1.9, 2.7, 0.6, 5.9, 1.6, 3.8, 1.4, 5.6},
  {1.4, 1.7, 0.9, 0.7, 0.8, 2.3, 1.2, 1.6, 0.8, 2.5},
  {1.6, 1.7, 0.9, 0.7, 0.3, 3.1, 1.0, 0.9, 0.6, 1.5},
  {-0.2, -0.5, -0.2, -0.1, 0.0, -0.4, 0.0, -0.5, -0.1, -0.6},
  {-0.1, -0.9, -0.8, -0.6, 0.0, -1.8, -0.6, -1.0, -0.5, -1.4},
  {-0.1, -0.6, -1.1, -1.2, -0.2, -5.4, -0.8, -1.0, -0.8, -1.1},
  {-0.5, -1.6, -2.0, -1.8, -0.9, -4.5, -1.8, -1.9, -1.0, -3.8},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.1, -0.2, -0.3, -0.2, 0.0, -0.5},
  {0.9, 3.1, 3.8, 3.9, 1.1, 12.7, 2.4, 4.7, 2.1, 7.5},
  {-0.3, -0.3, 0.2, -0.2, -0.1, -0.2, -0.2, -0.1, 0.2, -0.3},
  {0.0, -0.9, -1.3, -1.0, 0.0, -3.3, -0.9, -1.5, -0.8, -1.7},
  {-0.2, -0.7, -1.1, -1.3, -0.5, -5.3, -0.8, -1.4, -0.8, -1.7},
  {-0.4, -1.8, -1.9, -1.4, -0.7, -3.6, -2.0, -1.7, -0.9, -3.8},
  {0.2, 1.2, 1.4, 0.5, 0.7, 1.0, 1.4, 0.7, 0.7, 1.2},
  {0.5, 2.4, 2.6, 3.2, 0.3, 11.4, 2.0, 3.5, 1.5, 5.6},
  {-0.1, -1.0, -1.0, -0.6, 0.0, -1.9, -0.8, -1.2, -0.7, -1.3},
  {-0.1, -0.4, -0.6, -0.8, -0.1, -3.4, -0.5, -0.5, -0.4, -0.8},
  {-0.2, -0.9, -1.1, -1.4, -0.5, -5.1, -0.8, -1.4, -0.7, -2.2},
  {-0.4, -1.3, -1.8, -1.0, -0.7, -2.3, -2.1, -1.2, -0.9, -2.9},
  {0.4, 1.4, 2.4, 0.7, 1.1, 1.9, 2.5, 1.1, 1.3, 1.8},
  {0.6, 2.0, 1.7, 2.8, 0.2, 10.3, 1.2, 3.0, 1.0, 5.3},
  {0.1, -0.2, -0.2, -0.1, -0.1, -0.4, -0.2, -0.5, -0.2, -0.5},
  {0.0, -0.9, -1.5, -1.1, -0.2, -5.8, -1.5, -1.5, -1.2, -1.7},
  {0.0, -0.8, -1.8, -1.7, -0.8, -4.8, -1.8, -1.3, -1.5, -2.1},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
  {0.4, 1.6, 1.8, 1.3, 0.7, 3.4, 1.9, 1.6, 0.9, 3.5},
  {1.3, 1.7, 3.8, 3.9, 1.1, 9.8, 2.5, 5.0, 2.1, 7.0},
  {0.6, 2.0, 1.7, 2.5, 0.5, 7.0, 1.2, 3.0, 1.5, 4.8},
  {-0.5, -2.5, -2.4, -3.0, -0.7, -8.0, -2.5, -4.0, -2.0, -5.0},
  {-0.4, -2.0, -2.0, -1.8, -0.9, -4.8, -2.0, -2.0, -1.5, -3.9},
  {-0.4, -1.3, -1.8, -0.9, -0.6, -2.0, -2.0, -1.5, -1.2, -2.2},
  {-0.4, -0.9, -1.1, -0.8, -0.5, -1.0, -1.2, -1.3, -1.2, -1.7},
  {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
};

/* Target Patterns */
double  target_pat[PATTERNS][NUM_OUT] = {
  {0.0, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.05, 0.0},  {0.1, 0.0},
  {0.6, 0.0},  {0.6, 0.0},  {0.05, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.35, 0.0},
  {0.28, 0.0},  {0.6, 0.0},  {0.4, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.3, 0.0},
  {0.3, 0.0},  {0.75, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.35, 0.0},
  {0.2, 0.0},  {0.4, 0.0},  {0.3, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.0, 0.0},
  {0.3, 0.0},  {0.5, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.0, 0.85},  {0.0, 0.7},
  {0.0, 0.0},  {0.0, 0.0},  {0.85, 0.0},  {0.0, 0.0},  {0.0, 0.0},  {0.0, 0.8},
  {0.0, 0.85},  {0.0, 0.0},  {0.0, 0.0},  {0.0, 0.4},  {0.0, 0.4},  {0.0, 0.6},
```

```
  {0.0,0.85}, {0.0,0.0}, {0.0,0.0}, {0.0,0.5}, {0.0,0.3}, {0.0,0.6},
  {0.0,0.75}, {0.0,0.0}, {0.0,0.0}, {0.0,0.3}, {0.0,0.4}, {0.0,0.85},
  {0.0,0.0}, {0.0,0.0}, {0.0,0.0}, {0.3,0.0}, {0.75,0.0}, {0.5,0.0},
  {0.0,0.0}, {0.0,0.0}, {0.0,0.75}, {0.0,0.82},
};

int iterations = DEFAULT_ITER;
double out_f(), delta_f_out(), delta_f_hid(), rdm(), pattern_error();
double pattern_err[PATTERNS];

void
print_header()
{
  printf("%s%s",
         "¥n¥tHand Motion Interpretation Using a¥n",
         "¥t¥tSecond Order Recurrent Network¥n¥n");
}


/************************************
  create input, hidden, output units
************************************/
create_processing_units(pu, tampon1, tampon2)
struct unit *pu[];
double tampon1[], tampon2[];
{
  int   id;                             /* processing unit index */
  struct unit *create_unit();

  for (id = IN_UID(0); id < IN_UID(NUM_IN); id++)
    pu[id] = create_unit(id, "input", 0.0, NULL, 0.0, NULL, 0.0);
  for (id = CON_UID1(0); id < CON_UID1(NUM_CON1); id++)
    pu[id] = create_unit(id, "context1", 0.0, NULL, 0.0, NULL, 0.0);
  for (id = CON_UID2(0); id < CON_UID2(NUM_CON2); id++)
    pu[id] = create_unit(id, "context2", 0.0, NULL, 0.0, NULL, 0.0);
  for (id = HID_UID(0); id < HID_UID(NUM_HID); id++)
    pu[id] = create_unit(id, "hidden", 0.0, out_f, 0.0, delta_f_hid,rdm());
  for (id = OUT_UID(0); id < OUT_UID(NUM_OUT); id++)
    pu[id] = create_unit(id, "output", 0.0, out_f, 0.0, delta_f_out,rdm());
  for (id = 0; id < NUM_CON1; id++) {
    tampon1[id] = 0.5;                  /* initial values of context units */
    tampon2[id] = 0.5;
  }
}

/*********************************************
  create links - fully connected for each layer
*********************************************/
create_in_out_links(pu)
struct unit *pu[];
{
  int   i, j, k;                /* i == to and j == from unit id's */
  struct link *create_link();

  /* fully connected units */
  for (i = HID_UID(0); i < HID_UID(NUM_HID); i++) {     /* links to hidden */
    for (j = IN_UID(0); j < IN_UID(NUM_IN); j++)        /* from input units */
      pu[j]->outlinks =
        pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                     (char *)NULL, rdm(), 0.0);
    for (j = CON_UID1(0); j < CON_UID1(NUM_CON1); j++)  /* from context 1 units */
      pu[j]->outlinks =
        pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                     (char *)NULL, rdm(), 0.0);
    for (j = CON_UID2(0); j < CON_UID2(NUM_CON2); j++)  /* from context 2 units */
      pu[j]->outlinks =
        pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                     (char *)NULL, rdm(), 0.0);
  }
  for (i = OUT_UID(0); i < OUT_UID(NUM_OUT); i++) {     /* links to output */
    for (j = HID_UID(0); j < HID_UID(NUM_HID); j++)     /* from hidden units */
      pu[j]->outlinks =
        pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                     (char *)NULL, rdm(), 0.0);
  }
}

/*********************************************
  return a random number between -1.0 and 1.0
*********************************************/
double
rdm()
{
  return((rand() % 50000 - 25000)/ 25000.0);
}

struct unit *
create_unit(uid, label, output, out_f, delta, delta_f, thd)
int  uid;
char *label;
double   output, delta, thd;
double   (*out_f)(), (*delta_f)();
{
  struct unit  *unitptr;

  if (!(unitptr = (struct unit *)malloc(sizeof(struct unit)))) {
    fprintf(stderr, "create_unit: not enough memory¥n");
    exit(1);
  }
```

```
  /* initialize unit data */
  unitptr->uid = uid;
  unitptr->label = label;
  unitptr->output = output;
  unitptr->unit_out_f = out_f;          /* ptr to output function */
  unitptr->delta = delta;
  unitptr->unit_delta_f = delta_f;
  unitptr->threshold = thd;
  return (unitptr);
}

struct link *
create_link(start_inlist, to_uid, start_outlist, from_uid, label, wt, data)
struct link *start_inlist, *start_outlist;
int     to_uid, from_uid;
char *  label;
double  wt, data;
{
  struct link  *linkptr;

  if (!(linkptr = (struct link *)malloc(sizeof(struct link)))) {
    fprintf(stderr, "create_link: not enough memory¥n");
    exit(1);
  }
  /* initialize link data */
  linkptr->label = label;
  linkptr->from_unit = from_uid;
  linkptr->to_unit = to_uid;
  linkptr->weight = wt;
  linkptr->data = data;
  linkptr->next_inlink = start_inlist;
  linkptr->next_outlink = start_outlist;
  return(linkptr);
}

/*****************************************
  returns the first character of a command
*****************************************/
char
get_command(s)
char    *s;
{
  char  command[BUFSIZE];

  fputs(s, stdout);
  fflush(stdin); fflush(stdout);
  fgets(command, BUFSIZE, stdin);
  return((command[0]));         /* return 1st letter of command */
}

/*********************
  LEARNING ALGORITHM
*********************/
learn(pu, tampon1, tampon2)
struct unit *pu[];
double tampon1[], tampon2[];
{
  int j, k;
  register i, temp;
  char   tempstr[BUFSIZE];
  extern int    iterations;
  static char prompt[] = "Enter # iterations (default is 8000) => ";
  static char quote1[] = "I should do more learning.¥n";

  printf(prompt);
  fflush(stdin); fflush(stdout);
  gets(tempstr);
  if (temp = atoi(tempstr))
    iterations = temp;

  printf("¥nLearning ");
  for (i = 0; i < iterations; i++) {
    if ((i % NOTIFY) == 0) {
      printf(".");
      fflush(stdout);
    }
    if (i > iterations - PATTERNS)
      bp_learn(pu, tampon1, tampon2, 1);
    else
      bp_learn(pu, tampon1, tampon2, 0);
  }
  printf(" Done¥n¥n");
  for (j = 0; j < PATTERNS; j=j+2)
    printf("Error for pattern %d = ¥t%lf¥tError for pattern %d =¥t%lf¥n", j, pattern_err[j], j+1, pa
ttern_err[j+1]);

  for (j = 0; j < PATTERNS; j++) {
    if (pattern_err[j] > ERROR_TOLERANCE) {
      printf("¥nI don't know pattern %d very well.¥n%s",j, quote1);
      break;
    } else if (j == PATTERNS - 1) {
      printf("¥nOK. I know all the patterns quite well, now.¥n");
      break;
      }
  }
}

/***************************
  back propagation learning
```

```
**************************/
bp_learn(pu, tampon1, tampon2, save_error)
struct unit *pu[];
double tampon1[], tampon2[];
int    save_error;
{
  int i;
  static int count = 0;
  static int pattern = 0;
  extern int iterations;
  extern double pattern_err[PATTERNS];

  init_input_units(pu, pattern, tampon1, tampon2); /* initialize input pattern to
                                                      learn */
  propagate(pu, tampon1, tampon2);                /* calc outputs to check versus
                                                      targets */
  if (save_error)
    pattern_err[pattern] = pattern_error(pattern, pu);
  bp_adjust_weights(pattern, pu);
  if (pattern < PATTERNS - 1)
    pattern++;
  else
     pattern = 0;
  count++;
}

/*********************************************************************
  initialize the input units with a specific input pattern to learn
*********************************************************************/
init_input_units(pu, pattern, tampon1, tampon2)
struct unit *pu[];
int    pattern;
double tampon1[], tampon2[];
{
  int   id;

  for (id = IN_UID(0); id < IN_UID(NUM_IN); id++)
    pu[id]->output = input_pat[pattern][id];
  for (id = CON_UID1(0); id < CON_UID1(NUM_CON1); id++)
    pu[id]->output = tampon1[id - CON_UID1(0)];
  for (id = CON_UID2(0); id < CON_UID2(NUM_CON2); id++)
    pu[id]->output = tampon2[id - CON_UID2(0)];
}

/*******************************************
  calculate the activation level of each unit
*******************************************/
propagate(pu, tampon1, tampon2)
struct unit *pu[];
double tampon1[], tampon2[];
{
  int   id;

  for (id = HID_UID(0); id < HID_UID(NUM_HID); id++)
    (*(pu[id]->unit_out_f))(pu[id], pu);
  for (id = OUT_UID(0); id < OUT_UID(NUM_OUT); id++)
    (*(pu[id]->unit_out_f))(pu[id], pu);
  for (id = 0; id < NUM_CON1; id++)
    tampon2[id] = tampon1[id];
  for (id = HID_UID(0); id < HID_UID(NUM_HID); id++)
    tampon1[id - HID_UID(0)] = pu[id]->output;
}

/*********************************************************
  function to calculate the activation or output of units
*********************************************************/
double
out_f(pu_ptr, pu)
struct unit *pu_ptr, *pu[];
{
  double sum = 0.0, exp();
  struct link *tmp_ptr;

  tmp_ptr = pu_ptr->inlinks;
  while (tmp_ptr) {
    /* sum up (outputs from inlinks times weights on the inlinks) */
    sum += pu[tmp_ptr->from_unit]->output * tmp_ptr->weight;
    tmp_ptr = tmp_ptr->next_inlink;
  }
  sum += pu_ptr->threshold;
  pu_ptr->output = 1.0/(1.0 + exp(-sum));
}

/*****************************************************
  half of the sum of the squares of the errors of the
   output versus target values
*****************************************************/
double
pattern_error(pat_num, pu)
int     pat_num;                /* pattern number */
struct unit *pu[];
{
  int           i;
  double        temp, sum = 0.0;

  for (i = OUT_UID(0); i < OUT_UID(NUM_OUT); i++) {
    temp = target_pat[pat_num][TARGET_INDEX(i)] - pu[i]->output;
    sum += temp * temp;
  }
```

```
  return (sum/2.0);
}

/*********************
  weights adjustments
*********************/
bp_adjust_weights(pat_num, pu)
int     pat_num;                /* pattern number */
struct unit *pu[];
{
  int           i;              /* processing units id */
  double        temp1, temp2, delta, error_sum;
  struct link   *inlink_ptr, *outlink_ptr;

  /* calc deltas */
  for (i = OUT_UID(0); i < OUT_UID(NUM_OUT); i++) /* for each output unit, */
    (*(pu[i]->unit_delta_f))(pu, i, pat_num);     /* calculation of delta */
  for (i = HID_UID(0); i < HID_UID(NUM_HID); i++) /* for each hidden unit, */
    (*(pu[i]->unit_delta_f))(pu, i);              /* calculation of delta */
  /* calculate weights and thresholds */
  for (i = OUT_UID(0); i < OUT_UID(NUM_OUT); i++) {     /* for output units */
    inlink_ptr = pu[i]->inlinks;
    while (inlink_ptr) {        /* for each inlink to output unit */
      temp1 = LEARNING_RATE * pu[i]->delta *
        pu[inlink_ptr->from_unit]->output;
      temp2 = MOMENTUM * inlink_ptr->data;
      inlink_ptr->data = temp1 + temp2;           /* new delta weight */
      inlink_ptr->weight += inlink_ptr->data;     /* new weight */
      inlink_ptr = inlink_ptr->next_inlink;
    }
    pu[i]->threshold += LEARNING_RATE * pu[i]->delta;    /* new threshold */
  }
  for (i = HID_UID(0); i < HID_UID(NUM_HID); i++) {   /* for each hidden unit */
    inlink_ptr = pu[i]->inlinks;
    while (inlink_ptr) {        /* for each inlink to output unit */
      temp1 = LEARNING_RATE * pu[i]->delta *
        pu[inlink_ptr->from_unit]->output;
      temp2 = MOMENTUM * inlink_ptr->data;
      inlink_ptr->data = temp1 + temp2;               /* new delta weight */
      inlink_ptr->weight += inlink_ptr->data;         /* new weight */
        inlink_ptr = inlink_ptr->next_inlink;
    }
    pu[i]->threshold += LEARNING_RATE * pu[i]->delta;    /* new threshold */
  }
}

/***************************************
  calculate the delta for an output unit
***************************************/
double
delta_f_out(pu, uid, pat_num)
struct unit *pu[];
int    uid, pat_num;
{
  double        temp1, temp2, delta;

  /* calc deltas */
  temp1 = (target_pat[pat_num][TARGET_INDEX(uid)] - pu[uid]->output);
  temp2 = (1.0 - pu[uid]->output);
  delta = temp1 * pu[uid]->output * temp2; /* calc delta */
  pu[uid]->delta = delta; /* store delta to pass on */
}

/***************************************
  calculate the delta for a hidden unit
***************************************/
double
delta_f_hid(pu, uid)
struct unit *pu[];
int    uid;
{
  double        temp1, temp2, delta, error_sum;
  struct link   *inlink_ptr, *outlink_ptr;

  outlink_ptr = pu[uid]->outlinks;
  error_sum = 0.0;
  while (outlink_ptr) {
    error_sum += pu[outlink_ptr->to_unit]->delta * outlink_ptr->weight;
    outlink_ptr = outlink_ptr->next_outlink;
  }
  delta = pu[uid]->output * (1.0 - pu[uid]->output) * error_sum;
  pu[uid]->delta = delta;
}

/*****************************************************************************
  save weight of all the connections and threshold of hidden and output units
  in a file called 'WEIGHTS_FILE'
*****************************************************************************/
save_weights(pu)
struct unit *pu[];
{
  int i, j, k = 0, l = 0;
  double table_weights[(NUM_IN + NUM_CON1 + NUM_CON2)*NUM_HID + NUM_HID*NUM_OUT + NUM_HID + NUM_OUT]
;             /* size = total number of links + number of thresholds; only */
                        /* hidden and output units have a threshold                  */
  FILE *fiche;
  static char file_name[] = "WEIGHTS_FILE";
  struct link *tmp_ptr;
```

```
  for (i = HID_UID(0); i < HID_UID(NUM_HID); i++) {
    tmp_ptr = pu[i]->inlinks;
    table_weights[k*(NUM_IN + NUM_CON1 + NUM_CON2 + 1)] = pu[i]->threshold;
    j = (k + 1)*(NUM_IN + NUM_CON1 + NUM_CON2 + 1) - 1;
    while (tmp_ptr) {
        table_weights[j] = tmp_ptr->weight;
        j--;
        tmp_ptr = tmp_ptr->next_inlink;
    }
    k++;
  }

  for (i = OUT_UID(0); i < OUT_UID(NUM_OUT); i++) {
    tmp_ptr = pu[i]->inlinks;
    table_weights[k*(NUM_IN + NUM_CON1 + NUM_CON2 + 1) + l*(NUM_HID + 1)] = pu[i]->threshold;
    j = k*(NUM_IN + NUM_CON1 + NUM_CON2 + 1) + (l + 1)*(NUM_HID + 1) -1;
    while (tmp_ptr) {
      table_weights[j] = tmp_ptr->weight;
        j--;
        tmp_ptr = tmp_ptr->next_inlink;
    }
    l++;
  }

  fiche = fopen(file_name, "w");
  for (i = 0; i < (NUM_IN + NUM_CON1 + NUM_CON2)*NUM_HID + NUM_HID*NUM_OUT + NUM_HID + NUM_OUT; i++)

      fprintf(fiche, "%lf¥n", table_weights[i]);
  fclose(fiche);
}

/*****************************************************************************
  load the network with weights and thresholds from the file 'WEIGHTS_FILE'
*****************************************************************************/
load_weights(pu)
struct unit* pu[];
{
  int i, j;
  double x;
  FILE *fp;
  static char name[] = "WEIGHTS_FILE";

  if ((fp = fopen(name, "r")) == NULL) {
     fprintf(stderr, "¥nCan't load the file -[%s]¥n", name);
     exit(1);
  }

  for (i = HID_UID(0); i < HID_UID(NUM_HID); i++) {
   fscanf(fp, "%lf", &x);
   pu[i]->threshold = x;

   for (j = IN_UID(0); j < IN_UID(NUM_IN); j++) {
    fscanf(fp, "%lf", &x);
    pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                                   (char *)NULL, x, 0.0);
   }

   for (j = CON_UID1(0); j < CON_UID1(NUM_CON1); j++) {
    fscanf(fp, "%lf", &x);
    pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                                   (char *)NULL, x, 0.0);
   }

   for (j = CON_UID2(0); j < CON_UID2(NUM_CON2); j++) {
    fscanf(fp, "%lf", &x);
    pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                                   (char *)NULL, x, 0.0);
   }
  }

  for (i = OUT_UID(0); i < OUT_UID(NUM_OUT); i++) {
   fscanf(fp, "%lf", &x);
   pu[i]->threshold = x;

   for (j = HID_UID(0); j < HID_UID(NUM_HID); j++) {
    fscanf(fp, "%lf", &x);
    pu[j]->outlinks = pu[i]->inlinks = create_link(pu[i]->inlinks, i, pu[j]->outlinks, j,
                                                   (char *)NULL, x, 0.0);
   }
  }

  fclose(fp);
}

/*************************************************************
  detection of a neutral position : open hand or closed hand
*************************************************************/
int
freeze(tab)
double tab[NUM_IN];
{
static double max[NUM_IN] = {112.0, 100.0, 135.0, 66.0, 78.0, 122.0, 165.0, 80.0, 67.0, 208.0};
static double min[NUM_IN] = {100.0, 65.0, 80.0, 48.0, 60.0, 60.0, 105.0, 47.0, 48.0, 100.0};

  int i = 0, stop = 1;

  do {                                    /* to detect open hand position */
     stop &= (tab[i] >= max[i]);
```

```
      if (stop == 0)
         break;
      i++;
   } while (i < NUM_IN);

   if (stop == 1) return stop;
   stop = 1; i = 0;

   do {                                        /* to detect closed hand position */
      stop &= (tab[i] <= min[i]);
      if (stop == 0)
         break;
      i++;
   } while (i < NUM_IN);

   return stop;       /* if stop = 1 then a neutral position has been detected */
}
```

```
/*****************************************************************************************
************************************* dg2cmd.h ******************************************
*****                                                                                   *****
*****                              Send Command Define                                  *****
*****                                                                                   *****
*****************************************************************************************
*****************************************************************************************/

#define LeadingByte     0x24

#define Repeat60        0x41
#define Repeat30        0x42
#define OneShot         0x43
#define SystemID        0x44
#define EmptyBuffer     0x45
#define UserRead        0x46
#define UserIRQ         0x47
#define QueryBright     0x48
#define ColdReset       0x49
#define WarmReset       0x4A
#define MemAlloc        0x4B
#define DeltaSend       0x4C
#define SetBright       0x4D
#define SetDim          0x4E
#define FillBuffer      0x4F
#define LoadTable       0x50
#define AccessPolhemus  0x51
#define Angles          0x52
#define NumSensors      0x53
#define SetFeedback     0x54
#define QueryCutoff     0x55
#define SetCutOff       0x56
#define FlexValues      0x57
#define UserWrite       0x58
#define JointMap        0x59
```

```
/*****************************************************************************
************************************ dg2def.h ******************************************
*****                                                                                 *****
*****                          Constant Values Define                                 *****
*****                                                                                 *****
*****************************************************************************
*****************************************************************************/

#define numCommands     26
#define numsensors      10

#define hallbit         4
#define polbit          2
#define flexbit         1
#define fullflexbit     128

#define RepEnable       1
#define DeltaEnable     2
#define GetCalibTable   3
#define GetUserByte     4
#define GetFeedBack     5
#define GetJointMap     6
#define GetTables       7

#define polunit         0.001998352
#define polangle        0.00699432716

/**  65 inches divided by 32768  **/
#define POLUNIT         0.0050384521

/**  4 radians (in degree) divided by 32768  **/
#define POLANG          0.0069941137
```

```
/*****************************************************************************************
************************************ dg2ext.h ********************************************
*****                                                                                   *****
*****                          Extra Use Include & Define                               *****
*****                                                                                   *****
*****************************************************************************************
*****************************************************************************************/

/**  Include File  **/
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <malloc.h>
#include <math.h>

/**  Define  **/
#ifndef FALSE
#define FALSE           0
#endif
#ifndef TRUE
#define TRUE            1
#endif
```

```
/*****************************************************************************
************************************ dg2str.h ********************************
*****                                                                       *****
*****                        Definition of Data Structures                  *****
*****                                                                       *****
*****************************************************************************
*****************************************************************************/

/**  Finger Size  **/

#define PARAM_COUNT     3
#define FINGER_SIZE     5

/**  Hall Data  **/
typedef struct {
        int     index;
        int     middle;
        int     ring;
        int     pinkie;
} Hall;

/**  Polhemus Data  **/
typedef struct {
        float   x;
        float   y;
        float   z;
        float   yaw;
        float   pitch;
        float   roll;
} Polhemus;

/**  Flex Data (Inner Bright)  **/
typedef struct {
        int     finger[FINGER_SIZE];
        float   param[PARAM_COUNT][FINGER_SIZE];
} FlexInnerBright;

/**  Flex Data (Outer Bright)  **/
typedef struct {
        int     finger[FINGER_SIZE];
        float   param[PARAM_COUNT][FINGER_SIZE];
} FlexOuterBright;

/**  Flex Data (Inner Dim)  **/
typedef struct {
        int     finger[FINGER_SIZE];
        float   param[PARAM_COUNT][FINGER_SIZE];
} FlexInnerDim;

/**  Flex Data (Outer Dim)  **/
typedef struct {
        int     finger[FINGER_SIZE];
        float   param[PARAM_COUNT][FINGER_SIZE];
} FlexOuterDim;
```

```
/*****************************************************************************
************************************ dg2use.h ********************************
*****                                                                       *****
*****                    User's Values and Flags Definition                 *****
*****                                                                       *****
*****************************************************************************
*****************************************************************************/

/**  Define to Use System V Compiler  **/
#define USE_SYSTEM_V

/**  Get Data Test Status
 **   (define to Data Get from Data Glove
 **    undefine to Make Random Dummy Data)  **/
#define GLOVE_DATA_GET

/**  Data Glove Connect Serial Port Device  **/
#define DG2_DEV "/dev/ttyd1"

/* 1: ONESHOT, 2: REREAT30, 3: REREAT60 */
#define CONTROL 1

/**  Set Value for Data Glove2  **/
#define DEFAULT_BRIGHT  48
#define DEFAULT_DIM     60
#define DEFAULT_CUTOFF  255

/**  Get Data Status  **/
/*#define       HALL_ON*/
/*#define       POL_ON*/
#define FLEX_ON                                 /* FLEX_ON mode is set */
/*#define       DIM_ON*/

/**  Get Data Length  **/
#define SEND_LENGTH     50
#define REPLY_LENGTH    500
#define HALL_LENGTH     4
#define POL_LENGTH      6
#define FLEX_LENGTH     10

/**  Other Length  **/
#define NUMINP          1
#define REPLEN          99
#define RESET_RETURN    5

/**  Error Check Count Define  **/
#define TRYLIMIT        100
#define ERRLIMIT        256

/**  Data Write(Read) Status  **/
#define HALL_STAT       8
#define POL_STAT        4
#define FLEX_STAT       2
#define DIM_STAT        1

/**  Counts for Sample Data Get  **/
#define DATA_GET_CNT    100
#define SET_BRIGHT_CNT  128
#define ANGLE_GET_CNT   10

#define TEST_START_BRIGHT       0
#define TEST_CUT_BRIGHT 1

/**  Constant Variable Size  **/
#define FILENAME_SIZE   30
#define FGETS_SIZE      500
#define MAXFILENUM      100

/**  Use Calibrate Data File Name  **/
#define STRAIGHT_FILE   "Straight.dat"
#define BEND_FILE       "Bend.dat"
#define BRIGHT_FILE     "Bright.dat"
#define MINMAX_FILE     "MinMax.dat"
```

```
/*****************************************************************************
********************************** get_time.h ********************************
*****                                                                       *****
*****                          Get the System Time                          *****
*****                                                                       *****
*****************************************************************************
*****************************************************************************/

#include <stdio.h>
#include <sys/time.h>

static  double  get_double_time();
static  float   get_float_time();

static  double  get_double_time()
{
  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));
}

static  float  get_float_time()
{
  double  double_time;
  long    integer_time;

  double_time = get_double_time();
  integer_time = double_time / 10000;
  integer_time *= 10000;

  return((float)(double_time - integer_time));
}
```

```
/*****************************************************************************
************************************ tracker.h ******************************
*****                                                                       *****
*****                          Tracker Library Header File                  *****
*****                                                                       *****
*****************************************************************************
*****************************************************************************/

/**  Define to Use System V Compiler  **/
#define USE_SYSTEM_V

#include <stdio.h>
#include <fcntl.h>
#include <math.h>

/**  Used Serial Port Device  **/
#define TR_DEV  "/dev/ttyd3"

/**  Alignment Default File Name  **/
#define CALIB_FILE      "3space.dat"
```

```
/*****************************************************************************
************************************ tracker.c ******************************
*****                                                                       *****
*****                          Library for 3SPACE Isotrak                   *****
*****                                                                       *****
*****************************************************************************
*****************************************************************************/

#include "tracker.h"

#ifdef  USE_SYSTEM_V
#include <sys/termio.h>
#else
#include <sgtty.h>
#include <sys/ttydev.h>
#include <sys/ioctl.h>
#include <sys/file.h>
#endif

#define SCALE 3.0

static  int  fdi, fdo;

/**********************
  Initializing routine
**********************/
trinit()
{
#ifdef  USE_SYSTEM_V
  struct  termio  buf;
#else
  struct  sgttyb  buf;
#endif

  fdi = open(TR_DEV, O_RDONLY + O_NDELAY);
  fdo = open(TR_DEV, O_WRONLY);

  if (fdi == -1 || fdo == -1) return -1;

#ifdef  USE_SYSTEM_V
  ioctl(fdi, TCGETA, &buf);

#if     DEBUG
  fprintf(stdout,
          "%x %x %x %x¥n",buf.c_iflag,buf.c_oflag,buf.c_cflag,  buf.c_lflag);
#endif

  buf.c_iflag = IGNBRK + IGNPAR;
  buf.c_oflag = 0;
  buf.c_cflag = B19200 + CS8 + CSTOPB + CREAD + CLOCAL;
  buf.c_lflag = NOFLSH;
  buf.c_cc[0] = buf.c_cc[1] = buf.c_cc[2] = 0;
  buf.c_cc[3] = buf.c_cc[4] = buf.c_cc[5] = 0;

  ioctl(fdo, TCSETA, &buf);
  ioctl(fdi, TCSETA, &buf);
#else
  ioctl(fdi, TIOCGETP, &buf);

#if     DEBUG
  fprintf(stdout, "%x %x %x¥n", buf.sg_ispeed, buf.sg_ospeed, buf.sg_flags);
#endif

  buf.sg_ispeed = buf.sg_ospeed = B19200;
  buf.sg_flags = RAW + CBREAK;

  ioctl(fdo, TIOCSETP, &buf);
  ioctl(fdi, TIOCSETP, &buf);
#endif

  write(fdo, "u10¥r02,1¥r", 9);
  write(fdo, "H2,0,0,-1¥r", 10);
  write(fdo, "f", 1);

/*  trset(3);   Just for compatibility */

  return 0;
}

/*****************************
  Enables to get Euler angles
*****************************/
trroton()
{
  write(fdo, "02,4,1¥r", 7);
  sleep(1);
}

/*****************************************************
  Reads calibration file (CALIB_FILE == '3space.dat')
*****************************************************/
trcalib(s)
     char    *s;
{
  FILE    *fd, *fopen();
  char    buf[256];

  if ((fd = fopen(s, "r")) == NULL) return -1;
```

```
  while(fgets(buf, 255, fd) != NULL) {
    buf[strlen(buf)-1] = '¥r';
    write(fdo, buf, strlen(buf));
    sleep(1);
  }

  return 0;
}

/*********************************************
  Enables/disables to use magnetic source No. 2
*********************************************/
tr2off()
{
  if (fdo == 0) return -1;
  write(fdo, "l2¥r", 3);

  return 0;
}

/*********************************
  To reset coordinates alignment
*********************************/
trareset(source)
     int     source;
{
  char    obuf[16];

  sprintf(obuf, "R%1d¥r", source);
  write(fdo, obuf, 3);
}

/***************************************
  To fix the position of the hemisphere
***************************************/
trhemis(station, x, y, z)
     int     station;
     float   x, y, z;
{
  char    obuf[128];

#ifdef  USE_SYSTEM_V
  sprintf(obuf, "H%1d,%f,%f,%f¥r", station, x, y, z);
#else
  sprintf(obuf, "H%1d,%4.2f,%4.2f,%4.2f¥r", station, x, y, z);
#endif
  write(fdo, obuf, strlen(obuf));
}

/*************************************************************
  To define which magnetic source and sensors will be used
*************************************************************/
trset(station)
     int     station;   /*bit patter of station number*/
{
  unsigned  char  cbuf[256];
  unsigned  char  obuf[16];
  int     mask, n;

  write(fdo, "l¥r", 2);    /* get current status */
  for(n = 0; n < 60000; n++);   /* wait return data */
  for(n = 0; n < 13; n++)
    if (read(fdi, cbuf+n, 1) != 1) n--;
  for(n = mask = 1; n < 9; n++) {
    if (((station & mask) && (cbuf[n+2] == '0')) ||
        ((~station & mask) && (cbuf[n+2] == '1'))) {
      sprintf(obuf, "l%1d¥r", n);
      write(fdo, obuf, 3);
    }
    mask <<= 1;
  }
}

static  unsigned  char  dbuf[256];
static  int  datacount, count;

/*************************************
  Sends a command to get position data
*************************************/
trposin()
{
#ifdef  USE_SYSTEM_V
  ioctl(fdi, TCFLSH, 0);
#else
  ioctl(fdi, TIOCFLUSH, FREAD);
#endif
  datacount = count = 0;
  write(fdo, "P", 1);
}

/*****************************************************************************************
  Receives position data from the tracker (if less than 1 data record is got, returns 0 or
  a minus value) : TYPE 1
*****************************************************************************************/
trposrdr(x, y, z)
     double  *x, *y, *z;
{
  int     n;
  short   w;
```

```
  if ((n = read(fdi, dbuf+datacount, 11-datacount)) == 0) {
    count++;
    if (count > 1000) {
      n = datacount;
      datacount = count = 0;
      return -n;
    }
    return 0;
  }
  else {
    datacount += n;
    if (datacount != 11) return 0;
    n = dbuf[1] - '0';
    w = dbuf[4]*256 + dbuf[3];
    *x = w * (65.48 * 2.54 / 32767);
    w = dbuf[6]*256 + dbuf[5];
    *y = w * (65.48 * 2.54 / 32767);
    w = dbuf[8]*256 + dbuf[7];
    *z = w * (65.48 * 2.54 / 32767);
    datacount = count = 0;
    return n;
  }
}

/*****************************************************************************************
  Receives position data from the tracker (if less than 1 data record is got, waits until a
  complete one is got) : TYPE 2
*****************************************************************************************/
trposrd(x, y, z)
     double  *x, *y, *z;
{
  int     n = datacount, loop = count;
  short   w;

  while(n < 11 && loop < 1000) {
    n += read(fdi, dbuf+n, 11-n);
    loop++;
  }

  if (loop != 1000) {
    n = dbuf[1] - '0';
    w = dbuf[4]*256 + dbuf[3];
    *x = w * (65.48 * 2.54 / 32767);
    w = dbuf[6]*256 + dbuf[5];
    *y = w * (65.48 * 2.54 / 32767);
    w = dbuf[8]*256 + dbuf[7];
    *z = w * (65.48 * 2.54 / 32767);
    datacount = count = 0;
    return n;
  }
  else {
    datacount = count = 0;
    return -1;  /* time out */
  }
}

/*********************************************************************
  Receives position and orientation data from the tracker ; TYPE 1
*********************************************************************/
trposrotrdr(x, y, z, ra, re, rr)
     double  *x, *y, *z;
     double  *ra, *re, *rr;
{
  int     n;
  short   w;

  if ((n = read(fdi, dbuf+datacount, 17-datacount)) == 0) {
    count ++;
    if (count > 1000) {
      n = datacount;
      datacount = count = 0;
      return -n;
    }
    return 0;
  }
  else {
    datacount += n;
    if (datacount != 17) return 0;
    n = dbuf[1] - '0';
    w = dbuf[4]*256 + dbuf[3];
    *x = w * (65.48 * 2.54 / 32767);
    w = dbuf[6]*256 + dbuf[5];
    *y = w * (65.48 * 2.54 / 32767);
    w = dbuf[8]*256 + dbuf[7];
    *z = w * (65.48 * 2.54 / 32767);
    w = dbuf[10]*256 + dbuf[9];
    *ra = w * 720 / M_PI / 32767;
    w = dbuf[12]*256 + dbuf[11];
    *re = w * 720 / M_PI / 32767;
    w = dbuf[14]*256 + dbuf[13];
    *rr = w * 720 / M_PI / 32767;
    datacount = count = 0;
    return n;
  }
}

/*****************************************************************
  Receives position and orientation data from the tracker ; TYPE 2
```

```
*********************************************************************/
trposrotrd(x, y, z, ra, re, rr)
     double  *x, *y, *z;
     double  *ra, *re, *rr;
{
  int     n = datacount, m, loop = count;
  short   w;

  while(n < 17 && loop < 1000) {
    m = read(fdi, dbuf+n, 17-n);
    if (m >= 0) n += m;
    loop++;
  }
  if (loop != 1000) {
    n = dbuf[1] - '0';
    w = dbuf[4]*256 + dbuf[3];
    *x = w * (65.48 * 2.54 / 32767);
    w = dbuf[6]*256 + dbuf[5];
    *y = w * (65.48 * 2.54 / 32767);
    w = dbuf[8]*256 + dbuf[7];
    *z = w * (65.48 * 2.54 / 32767);
    w = dbuf[10]*256 + dbuf[9];
    *ra = w * 720 / M_PI / 32767;
    w = dbuf[12]*256 + dbuf[11];
    *re = w * 720 / M_PI / 32767;
    w = dbuf[14]*256 + dbuf[13];
    *rr = w * 720 / M_PI / 32767;
    datacount = count = 0;
    return n;
  }
  else {
#if DEBUG
    for(loop = 0; loop < 17; loop++)
      fprintf(stdout, "%2x ", (int)dbuf[loop]);
    fprintf(stdout, "¥n");
#endif
    datacount = count = 0;
    return -n;  /* time out */
  }
}

/*****************************************************************************************
  Sends the tracker a command to set continuous transmission mode. Reads continuously n data
  records and stores them in a buffer.
*****************************************************************************************/
trcrawd(buf, n)
     char    *buf;
     int     n;
{
#ifdef  USE_SYSTEM_V
  ioctl(fdi, TCFLSH, 0);
#else
  ioctl(fdi, TIOCFLUSH, FREAD);
#endif

  write(fdo, "C", 1);

  for(n += 1; n; ) {
    if (read(fdi, buf, 1) == 0) continue;
    if (*buf++ & 0x80) n--;
  }

  write(fdo, "c", 1);

#ifdef  USE_SYSTEM_V
  ioctl(fdi, TCFLSH, 0);
#else
  ioctl(fdi, TIOCFLUSH, FREAD);
#endif
}

/*************************************************************
  Converts binary type data into coordinates and Euler angles
*************************************************************/
trdcposrot(buf, n, x, y, z, ra, re, rr)
     char    *buf;
     int     *n;
     double  *x, *y, *z;
     double  *ra, *re, *rr;
{
  char    dbuf[17];
  int     i = 0, j, k = 0, l;
  short   w;

  if (!(buf[k] & 0x80)) return -1;

  do {
    for(j = 0; j < 7; j++) {
      dbuf[i+j] = buf[k++];
      if (buf[k] & 0x80) {
        k--;
        break;
      }
    }
    for(l = 0; l < j; l++) {
      dbuf[i+l] = ((buf[k] & (0x01 << l)) ? dbuf[i+l] | 0x80 :
                   dbuf[i+l] & 0x7f);
    }
    i += j;
```

```
    k++;
  } while(!(buf[k] & 0x80));

  *n = dbuf[1] - '0';
  w = dbuf[4]*256 + dbuf[3];
  *x = w * (65.48 * 2.54 / 32767);
  w = dbuf[6]*256 + dbuf[5];
  *y = w * (65.48 * 2.54 / 32767);
  w = dbuf[8]*256 + dbuf[7];
  *z = w * (65.48 * 2.54 / 32767);
  w = dbuf[10]*256 + dbuf[9];
  *ra = w * 720 / M_PI / 32767;
  w = dbuf[12]*256 + dbuf[11];
  *re = w * 720 / M_PI / 32767;
  w = dbuf[14]*256 + dbuf[13];
  *rr = w * 720 / M_PI / 32767;

  return k;
}
```

```
/*****************************************************************************************
************************************ double_get.c ***************************************
*****                                                                                   *****
*****          Here are defined some functions concerning the DataGlove.                *****
*****                                                                                   *****
*****************************************************************************************
*****************************************************************************************/

#include "dg2cmd.h"
#include "dg2def.h"
#include "dg2ext.h"
#include "dg2str.h"
#include "dg2use.h"
#include "tracker.h"
#include "get_time.h"
#include <sys/types.h>
#include <fcntl.h>

/**  only station No.7 is active  **/
#define ACTIVE_STATION  64

FILE *file_name_check();

/*********************************************************
  initialization of the DataGlove and the 3Space Isotrak
*********************************************************/
int  double_init()
{
  FILE    *fp;
  int     set_bright, set_dim;
  int     n, m = ACTIVE_STATION, loop, mask;

  fprintf(stderr, "DataGlove2 Setup & Reset ...");
  fflush(stderr);
  if (OpenModem() == FALSE) exit(-1);
  while(!Cold_Reset());

  /**  Data Glove Model 2: to load bright level file  **/
  /**  BRIGHT_FILE == 'Bright.datxx' where xx is a two digit number  **/
  if ((fp = file_name_check(BRIGHT_FILE)) != NULL) {
    fscanf(fp, "%d", &set_bright);
    set_dim = set_bright;
    Set_Bright(set_bright, set_dim);
    fclose(fp);
  }
  else {
    fprintf(stderr,
            "¥n Warning : Can't Load Bright Level File  - [%s] ...",
            BRIGHT_FILE);
    fflush(stderr);
  }
  fprintf(stderr, " Completed¥n¥n");
  fflush(stderr);

  fprintf(stderr, "3Space Tracker Setup ...");
  fflush(stderr);
  fprintf(stderr, "¥n Get Station No. ");
  fflush(stderr);
  for(loop = 0, mask = n = 1; n < 9; mask <<= 1, n++) {
    if (mask & m) {
      fprintf(stderr, "[%d] ", n);
      fflush(stderr);
      loop++;
    }
  }
  fprintf(stderr, " ...");
  fflush(stderr);

  /**  3Space Tracker: to load calibration file  **/
  /**  CALIB_FILE == '3space.dat'                **/
  if ((fp = fopen(CALIB_FILE, "r")) != NULL) {
    fclose(fp);
    fprintf(stderr, "¥n 3Space Tracker Calibration ...");
    fflush(stderr);
    if (trinit() == -1) {
      fprintf(stderr, "Error : Can't Initialize 3Space Tracker¥n");
      exit(-1);
    }

    trcalib(CALIB_FILE);
    fprintf(stderr, " Completed¥n");
    fflush(stderr);
  }
  else {
    fprintf(stderr,
            "¥n Warning : Can't Load Calibration File - [%s] ...", CALIB_FILE);
    fflush(stderr);
  }

  fprintf(stderr, "¥n 3Space Tracker Station Set ...");
  if (trinit() == -1) {
    fprintf(stderr, "Error : Can't Initialize 3Space Tracker¥n");
    exit(-1);
  }
  trset(m);
  trroton();
  sleep(1);
  fprintf(stderr, " Completed¥n¥n");
  fflush(stderr);
```

```
  return(loop);
}

/*****************************************************************************
  checks if a file has "namexx" format (where xx is a two digit number) and selects
  the file with the greatest number xx.
*****************************************************************************/
FILE  *file_name_check(fname)
     char    fname[];
{
  FILE    *fp, *cfp;
  char    new_fname[FILENAME_SIZE];
  int     fnum;

  for(fnum = 0; fnum < MAXFILENUM; fnum++) {
    sprintf(new_fname, "%s%02d", fname, fnum);
    if ((cfp = fopen(new_fname, "r")) == NULL)
      return((fnum != 0 ? fp : NULL));
    else {
      fclose(cfp);
      fp = fopen(new_fname, "r");
    }
  }
}
```

```
/*****************************************************************************************
********************************** dg2_operate.c ****************************************
*****                                                                                   *****
*****            Here are procedures concerning the DataGlove Model 2 :                 *****
*****   Commands to send & receive data                                                 *****
*****   ( Sun 3/260   <--> Data Glove Model 2 )    using Unix 4.2 BSD  Compiler         *****
*****   ( Iris4D/70GT <--> Data Glove Model 2 )    using Unix System V Compiler         *****
*****                                                                                   *****
*****************************************************************************************
*****************************************************************************************/

#include "dg2cmd.h"
#include "dg2def.h"
#include "dg2ext.h"
#include "dg2str.h"
#include "dg2use.h"

#ifdef  USE_SYSTEM_V
#include <sys/termio.h>
#include <fcntl.h>
#else
#include <sgtty.h>
#include <sys/ttydev.h>
#include <sys/ioctl.h>
#endif

#ifdef  GLOVE_DATA_GET
#include "get_time.h"
#else
#include "random.h"
#endif

/**  Cold Reset Use Time Minimum  **/
#define COLD_RESET_TIME 1.0

/**  Send Command in this String  **/
char    sendbytes[SEND_LENGTH];
int     replyenable, cinlen;
short   chall, cpol, cflex, cdim;

/*****************************************
  Get One Shot Data from Data Glove Model 2
*****************************************/
Get_OneShot(hl, ph, fib, fob, fid, fod)
     Hall               *hl;
     Polhemus           *ph;
     FlexInnerBright    *fib;        /* I use fib and fob, ie. the FLEX_ON mode */
     FlexOuterBright    *fob;
     FlexInnerDim       *fid;
     FlexOuterDim       *fod;
{
  unsigned char reply[REPLY_LENGTH];
  int   newenable = 0;
#ifdef  GLOVE_DATA_GET
  int   try, get_start, get_error, getcount, trycount;
  int   try_limit = TRYLIMIT, err_limit = ERRLIMIT;
  char  gch;
#else
  int   getcount;
#endif
  short data_hall[HALL_LENGTH];
  float data_pol[POL_LENGTH];
  short data_bright[FLEX_LENGTH], data_dim[FLEX_LENGTH];

#ifdef  GLOVE_DATA_GET
  replyenable = 0;

  /**  Device Enable Set  **/
#ifdef  HALL_ON
  newenable |= hallbit;
#endif
#ifdef  POL_ON
  newenable |= polbit;
#endif
#ifdef  FLEX_ON
  newenable |= flexbit;
#endif
#ifdef  DIM_ON
  newenable |= fullflexbit;
#endif
  replyenable = newenable;
  sendbytes[2] = replyenable;

  Predict(&cinlen, &chall, &cpol, &cflex, &cdim);

  /**  Send Command Set  **/
  sendbytes[0] = LeadingByte;
  sendbytes[1] = OneShot;

  /**  Receive Data from Data Glove Model 2  **/
  for(try = TRUE, trycount = 0; try && trycount < try_limit; trycount++) {
    SendString(sendbytes, NUMINP+2);
    getcount = get_error = 0;
    get_start = FALSE;
    while(getcount < cinlen) {
      if (Getbyte(&gch)) {
        get_start = TRUE;
        reply[getcount++] = gch;
      }
```

```
      else {
        if (get_start) {
          get_error++;
          /* continue roop by error case */
          if (get_error > err_limit) break;
        }
      }
    }
    /* cinlen is received data length FALSE to continue loop */
    if (getcount == cinlen) try = FALSE;
  }

  /**  Over Count Try receive (error)  **/
  if (try) {
    fprintf(stderr, "Error : Try Count Greater than %d times !!¥n", TRYLIMIT);
    exit();
  }

  reply[getcount] = '¥0';

  /**  Change Received String Data to Polhemus etc. Data **/
  RepData(reply, data_hall, data_pol, data_bright, data_dim,
          chall, cpol, cflex, cdim);

#else
  reply[0] = '¥0';
  for(getcount = 0; getcount < FLEX_LENGTH; getcount++) {
    if (getcount < HALL_LENGTH) data_hall[getcount] = get_random(256);
    if (getcount < POL_LENGTH) data_pol[getcount] = get_random(256);
    data_bright[getcount] = get_random(256);
    data_dim[getcount] = get_random(256);
  }
#endif

  /**  Set Data for Struct Area in Call Functions  **/
#ifdef  HALL_ON
  hl->finger[1] = data_hall[0]; hl->finger[2] = data_hall[1];
  hl->finger[3] = data_hall[2]; hl->finger[4] = data_hall[3];
#endif

#ifdef  POL_ON
  ph->x = data_pol[0];          ph->y = data_pol[1];
  ph->z = data_pol[2];          ph->yaw = data_pol[3];
  ph->pitch = data_pol[4];      ph->roll = data_pol[5];
#endif

#ifdef  FLEX_ON
  fib->finger[0] = data_bright[0];      fib->finger[1] = data_bright[2];
  fib->finger[2] = data_bright[4];      fib->finger[3] = data_bright[6];
  fib->finger[4] = data_bright[8];

  fob->finger[0] = data_bright[1];      fob->finger[1] = data_bright[3];
  fob->finger[2] = data_bright[5];      fob->finger[3] = data_bright[7];
  fob->finger[4] = data_bright[9];

#ifdef  DIM_ON
  fid->finger[0] = data_dim[0]; fid->finger[1] = data_dim[2];
  fid->finger[2] = data_dim[4]; fid->finger[3] = data_dim[6];
  fid->finger[4] = data_dim[8];

  fod->finger[0] = data_dim[1]; fod->finger[1] = data_dim[3];
  fod->finger[2] = data_dim[5]; fod->finger[3] = data_dim[7];
  fod->finger[4] = data_dim[9];
#endif
#endif


#ifdef  GLOVE_DATA_GET
  return(strlen(reply));
#else
  return(64);
#endif
}

/*********************************************************
  Set Repeat60 or Repeat30 Data from Data Glove Model 2
*********************************************************/
Set_Repeat(helts)
     short   helts;
{
#ifdef  GLOVE_DATA_GET
  int   newenable = 0;

  replyenable = 0;

  /**  Device Enable Set  **/
#ifdef  HALL_ON
  newenable |= hallbit;
#endif
#ifdef  POL_ON
  newenable |= polbit;
#endif
#ifdef  FLEX_ON
  newenable |= flexbit;
#endif
#ifdef  DIM_ON
  newenable |= fullflexbit;
#endif
  replyenable = newenable;
```

```
  sendbytes[2] = replyenable;

  Predict(&cinlen, &chall, &cpol, &cflex, &cdim);

  /**  Send Command Set  **/
  sendbytes[0] = LeadingByte;
  sendbytes[1] = (helts == 30 ? Repeat30 : Repeat60);
  SendString(sendbytes, NUMINP+2);
#endif
}

/*********************************************************
  Get Repeat60 or Repeat30 Data from Data Glove Model 2
*********************************************************/
Get_Repeat(hl, ph, fib, fob, fid, fod)
     Hall               *hl;
     Polhemus           *ph;
     FlexInnerBright    *fib;
     FlexOuterBright    *fob;
     FlexInnerDim       *fid;
     FlexOuterDim       *fod;
{
  unsigned char reply[REPLY_LENGTH];
#ifdef  GLOVE_DATA_GET
  int   try, get_start, get_error, getcount, trycount;
  int   try_limit = TRYLIMIT, err_limit = ERRLIMIT;
  char  gch;
#else
  int   getcount;
#endif
  short data_hall[HALL_LENGTH];
  float data_pol[POL_LENGTH];
  short data_bright[FLEX_LENGTH], data_dim[FLEX_LENGTH];

#ifdef  GLOVE_DATA_GET
  /**  Receive Data from Data Glove 2  **/
  for(try = TRUE, trycount = 0; try && trycount < try_limit; trycount++) {
    getcount = get_error = 0;
    get_start = FALSE;
    while(getcount < cinlen) {
      if (Getbyte(&gch)) {
        get_start = TRUE;
        reply[getcount++] = gch;
      }
      else {
        if (get_start) {
          get_error++;
          /* continue roop by error case */
          if (get_error > err_limit) break;
        }
      }
    }
    /* cinlen is received data length FALSE to continue loop */
    if (getcount == cinlen) try = FALSE;
  }

  /**  Over Count Try receive (error)  **/
  if (try) {
    fprintf(stderr, "Error : Try Count Greater than %d times !!¥n", TRYLIMIT);
    exit();
  }

  reply[getcount] = '¥0';

  /**  Change Received String Data to Polhemus etc. Data **/
  RepData(reply, data_hall, data_pol, data_bright, data_dim,
          chall, cpol, cflex, cdim);

#else
  reply[0] = '¥0';
  for(getcount = 0; getcount < FLEX_LENGTH; getcount++) {
    if (getcount < HALL_LENGTH) data_hall[getcount] = get_random(256);
    if (getcount < POL_LENGTH) data_pol[getcount] = get_random(256);
    data_bright[getcount] = get_random(256);
    data_dim[getcount] = get_random(256);
  }
#endif

  /**  Set Data for Struct Area in Call Functions  **/
#ifdef  HALL_ON
  hl->finger[1] = data_hall[0]; hl->finger[2] = data_hall[1];
  hl->finger[3] = data_hall[2]; hl->finger[4] = data_hall[3];
#endif

#ifdef  POL_ON
  ph->x = data_pol[0];          ph->y = data_pol[1];
  ph->z = data_pol[2];          ph->yaw = data_pol[3];
  ph->pitch = data_pol[4];      ph->roll = data_pol[5];
#endif

#ifdef  FLEX_ON
  fib->finger[0] = data_bright[0];      fib->finger[1] = data_bright[2];
  fib->finger[2] = data_bright[4];      fib->finger[3] = data_bright[6];
  fib->finger[4] = data_bright[8];

  fob->finger[0] = data_bright[1];      fob->finger[1] = data_bright[3];
  fob->finger[2] = data_bright[5];      fob->finger[3] = data_bright[7];
  fob->finger[4] = data_bright[9];
```

```
#ifdef  DIM_ON
  fid->finger[0] = data_dim[0]; fid->finger[1] = data_dim[2];
  fid->finger[2] = data_dim[4]; fid->finger[3] = data_dim[6];
  fid->finger[4] = data_dim[8];

  fod->finger[0] = data_dim[1]; fod->finger[1] = data_dim[3];
  fod->finger[2] = data_dim[5]; fod->finger[3] = data_dim[7];
  fod->finger[4] = data_dim[9];
#endif
#endif

#ifdef  GLOVE_DATA_GET
  return(strlen(reply));
#else
  return(64);
#endif
}

/*************************************************
  To Set the Get Data Flag and Get Data Length Set
*************************************************/
Predict(inlen, hall, pol, flex, dim)
     int        *inlen;
     short      *hall, *pol, *flex, *dim;
{
  *hall = ((replyenable & 0x04) ? TRUE : FALSE);
  *pol = ((replyenable & 0x02) ? TRUE : FALSE);
  *flex = ((replyenable & 0x01) ? TRUE : FALSE);
  *dim = ((replyenable & 0x80) ? TRUE : FALSE);

  *inlen = 4 + HALL_LENGTH*(*hall) +
    2*POL_LENGTH*(*pol) + FLEX_LENGTH*(1+*dim)*(*flex);
}

/*************************************************************
  Change Received String Data to Hall & Plhemus & Flex Data
*************************************************************/
RepData(reply, data_hall, data_pol, data_bright, data_dim,
        hall, pol, flex, dim)
     unsigned   char    reply[];
     short      data_hall[];
     float      data_pol[];
     short      data_bright[], data_dim[];
     short      hall, pol, flex, dim;
{
  int   setcnt, idx, start_point, end_point = 4;
  short lowbyte, hibyte, value;
  double  conscm, consrd;

  /**  Received Hall Data (One Data is 1 Byte)  **/
  if (hall) {
    start_point = end_point;
    end_point = start_point + 4;
    for(setcnt = 0, idx = start_point; idx < end_point; setcnt++, idx++) {
      data_hall[setcnt] = (short)reply[idx];
    }
  }

  /**  Received Polhemus Data (One Data is 2 Bytes)  **/
  if (pol) {
    start_point = end_point;
    end_point = start_point + 12;
    /**  Change Unit Value (Length & Angle)  **/
    conscm = POLUNIT;
    consrd = POLANG;
    for(setcnt = 0, idx = start_point; idx < end_point; setcnt++, idx += 2) {
      lowbyte = (int)reply[idx];
      hibyte = (int)reply[idx+1];
      hibyte <<= 8;
      value = hibyte | lowbyte;
      data_pol[setcnt] = (setcnt < 3 ? value * conscm : value * consrd);
    }
  }

  /**  Received Flex Data (One Data is 1 Byte)  **/
  if (flex) {
    start_point = end_point;
    end_point += (dim ? 20 : 10);
    for(setcnt = 0, idx = start_point; idx < end_point; setcnt++, idx++) {
      if (setcnt < 10)
        data_bright[setcnt] = (short)reply[idx];
      else
        data_dim[setcnt-10] = (short)reply[idx];
    }
  }
}

/*************************************************************
  To End Repeat60 or Repeat30 Data Get etc. Use WarmReset Only
*************************************************************/
Warm_Reset()
{
#ifdef  GLOVE_DATA_GET
  int   getcount;
  char  gch;

  /**  Warm Reset **/
  sendbytes[0] = LeadingByte;
  sendbytes[1] = WarmReset;
```

```
  SendString(sendbytes, 2);

  for(getcount = 0; getcount < RESET_RETURN; ) {
    if (Getbyte(&gch)) getcount++;
  }
#endif
}

/*****************
  Cold Reset Only
******************/
Cold_Reset()
{
#ifdef  GLOVE_DATA_GET
  int   getcount;
  char  gch;
  double  st;

  /**  Cold Reset **/
  st = get_double_time();

  sendbytes[0] = LeadingByte;
  sendbytes[1] = ColdReset;
  SendString(sendbytes, 2);

  for(getcount = 0; getcount < RESET_RETURN; ) {
    if (Getbyte(&gch)) getcount++;
  }

  return((get_double_time()-st > COLD_RESET_TIME ? 1 : 0));
#else
  return(1);
#endif
}


/*********************************
  Set Brightness & Dimness Level
*********************************/
Set_Bright(bright_level, dim_level)
     short      bright_level, dim_level;
{
#ifdef  GLOVE_DATA_GET
  sendbytes[0] = LeadingByte;
  sendbytes[1] = SetBright;
  sendbytes[2] = bright_level;
  SendString(sendbytes, 3);

#ifdef  DIM_ON
  sendbytes[1] = SetDim;
  sendbytes[2] = dim_level;
  SendString(sendbytes, 3);
#endif
#endif
}

/******************
  Set Cutoff Value
******************/
Set_Cutoff(cutoff_value)
     short      cutoff_value;
{
#ifdef  GLOVE_DATA_GET
  sendbytes[0] = LeadingByte;
  sendbytes[1] = SetCutOff;
  sendbytes[2] = cutoff_value;
  SendString(sendbytes, 3);
#endif
}

/*******************************************************
  Direct Send Command for Magnetic Sensor (Now Unused)
*******************************************************/
SendPolhemus(polmsg)
     char       *polmsg;
{
  int   sendlen = strlen(polmsg), polidx;

  /**  Send Command Must be Not Greater than 63 bytes  **/
  if (sendlen > 63) {
    fprintf(stderr, "Send Polhemus Command Length Greater Than 63 bytes !!¥n");
    exit(1);
  }

  /**  Send Command Set  **/
  sendbytes[0] = LeadingByte;
  sendbytes[1] = AccessPolhemus;
  sendbytes[2] = sendlen;

  for(polidx = 0; polmsg[polidx] != '¥0'; polidx++) {
    sendbytes[polidx+3] = polmsg[polidx];
  }

  fprintf(stderr, "Send Length  = %d¥nSend Message = %s¥n", sendlen, polmsg);

  sendbytes[66] = '¥0';
  SendString(sendbytes, 66);
}
```

```
/*************************************
  Open Serial Port & Configuration Set
    Suport Unix 4.2 BSD & System V
*************************************/
static  int     fdi, fdo;

OpenModem()
{
#ifdef  GLOVE_DATA_GET

#ifdef  USE_SYSTEM_V
  struct        termio  buf;
#else
  struct        sgttyb  buf;
#endif

  /**  Serial Port Open  **/
  fdi = open(DG2_DEV, O_RDONLY | O_NDELAY);
  fdo = open(DG2_DEV, O_WRONLY);

  if (fdi == -1 || fdo == -1) {
    fprintf(stderr, "Error : Can't Open Device '%s'¥n", DG2_DEV);
    return FALSE;
  }

  /**  Speed = 19200 ; No Parity Check ; 1 Stop Bits  **/
#ifdef  USE_SYSTEM_V
  ioctl(fdi, TCGETA, &buf);

  buf.c_iflag  = IGNBRK + IGNPAR;
  buf.c_oflag  = 0;
  buf.c_cflag  = B19200 + CS8 + CREAD + CSTOPB + CLOCAL;
  buf.c_lflag  = NOFLSH;
  buf.c_cc[0] = buf.c_cc[1] = buf.c_cc[2] = 0;
  buf.c_cc[3] = buf.c_cc[4] = buf.c_cc[5] = 0;

  ioctl(fdo, TCSETA, &buf);
  ioctl(fdi, TCSETA, &buf);
#else
  ioctl(fdi, TIOCGETP, &buf);

  buf.sg_ispeed = buf.sg_ospeed = B19200;
  buf.sg_flags  = RAW + CBREAK;

  ioctl(fdo, TIOCSETP, &buf);
  ioctl(fdi, TIOCSETP, &buf);
#endif

#else
  set_random();
#endif
  return TRUE;
}

/*****************************************
  Get Byte from Serial port (Not Buffuring)
*****************************************/
Getbyte(ch)
     char       *ch;
{
  int     stat;

  while((stat = read(fdi, ch, 1)) != 1) {
    /**  Read Error  **/
    if (stat == -1) exit(1);
  }

  return(1);
}

/*****************************************
  Send String to Serial port (Not Buffuring)
*****************************************/
SendString(outstring, length)
     char       *outstring;
     int        length;
{
  return write(fdo, outstring, (unsigned)length);
}
```

```
/*****************************************************************************
******************************** dg2_calibrate.c ****************************
*****                                                                       *****
*****                  This program is used to calibrate the DataGlove.     *****
*****    The calibration is user dependent. It creates the following files :  *****
*****    STRAIGHT_FILE (data for fully open hand); BEND_FILE (first for thumb bent  *****
*****    and then for the 4 fingers bent); with these 2 files, BRIGHT_FILE (to set  *****
*****    the level of light intensity through the optical fibers) is created.  *****
*****                                                                       *****
*****************************************************************************
*****************************************************************************/

#include "dg2cmd.h"
#include "dg2def.h"
#include "dg2ext.h"
#include "dg2str.h"
#include "dg2use.h"

#define INVALID_THRESH  10

static  void    bright_level_set();
static  FILE    *file_name_create();

#ifdef  FLEX_ON
static  FlexInnerBright straight_in[SET_BRIGHT_CNT], bend_in[SET_BRIGHT_CNT];
static  FlexOuterBright straight_out[SET_BRIGHT_CNT], bend_out[SET_BRIGHT_CNT];
#endif

FILE    *fopen();

void  main(argc, argv)
     int     argc;
     char    **argv;
{
  static        Hall            hl;
  static        Polhemus        ph;
  static        FlexInnerBright fib;
  static        FlexOuterBright fob;
  static        FlexInnerDim    fid;
  static        FlexOuterDim    fod;
  FILE    *fp;
  int     get_cnt, read_length;
  register int  idx, fin_idx;
  int     set_bright, set_dim;

  if (OpenModem() == FALSE) exit(-1);
  while(!Cold_Reset());

  fprintf(stderr, "Now First Calibration...¥n");
  fprintf(stderr, "Set All Fingers Straight on the Table¥n");
  fprintf(stderr, "Ready ? Press Return-Key : ");
  getchar();

#if     CONTROL == 2
  Set_Repeat(30);
#endif
#if     CONTROL == 3
  Set_Repeat(60);
#endif

  for(set_bright = 0; set_bright < SET_BRIGHT_CNT; set_bright++) {
    Warm_Reset();
    Set_Bright(set_bright, set_bright);
#if     CONTROL == 1
    read_length = Get_OneShot(&hl, &ph, &fib, &fob, &fid, &fod);
#else
    read_length = Get_Repeat(&hl, &ph, &fib, &fob, &fid, &fod);
#endif

#ifdef  FLEX_ON
    for(fin_idx = 0; fin_idx < FINGER_SIZE; fin_idx++) {
      straight_in[set_bright].finger[fin_idx] = fib.finger[fin_idx];
      straight_out[set_bright].finger[fin_idx] = fob.finger[fin_idx];
    }
#endif
  }
  Warm_Reset();

  fprintf(stderr, "Now Second Calibration...¥n");
  fprintf(stderr, "Set Thumb Inner - 45 deg and Outer - 90 deg¥n");
  fprintf(stderr, "Ready ? Press Return-Key : ");
  getchar();

#if     CONTROL == 2
  Set_Repeat(30);
#endif
#if     CONTROL == 3
  Set_Repeat(60);
#endif

  for(set_bright = 0; set_bright < SET_BRIGHT_CNT; set_bright++) {
    Warm_Reset();
    Set_Bright(set_bright, set_bright);
#if     CONTROL == 1
    read_length = Get_OneShot(&hl, &ph, &fib, &fob, &fid, &fod);
#else
    read_length = Get_Repeat(&hl, &ph, &fib, &fob, &fid, &fod);
#endif
```

```
#ifdef  FLEX_ON
    bend_in[set_bright].finger[0] = fib.finger[0];
    bend_out[set_bright].finger[0] = fob.finger[0];
#endif
  }
  Warm_Reset();

  fprintf(stderr, "Now Third Calibration...¥n");
  fprintf(stderr, "Set Fingers except Thumb, Inner - 90 deg and Outer - 90 deg¥n");
  fprintf(stderr, "Ready ? Press Return-Key : ");
  getchar();

#if     CONTROL == 2
  Set_Repeat(30);
#endif
#if     CONTROL == 3
  Set_Repeat(60);
#endif

  for(set_bright = 0; set_bright < SET_BRIGHT_CNT; set_bright++) {
    Warm_Reset();
    Set_Bright(set_bright, set_bright);
#if     CONTROL == 1
    read_length = Get_OneShot(&hl, &ph, &fib, &fob, &fid, &fod);
#else
    read_length = Get_Repeat(&hl, &ph, &fib, &fob, &fid, &fod);
#endif

#ifdef  FLEX_ON
    for(fin_idx = 1; fin_idx < FINGER_SIZE; fin_idx++) {
      bend_in[set_bright].finger[fin_idx] = fib.finger[fin_idx];
      bend_out[set_bright].finger[fin_idx] = fob.finger[fin_idx];
    }
#endif
  }

  fp = file_name_create(STRAIGHT_FILE);
  for(set_bright = 0; set_bright < SET_BRIGHT_CNT; set_bright++) {
#ifdef  FLEX_ON
    fprintf(fp, "%3d ", set_bright);
    for(fin_idx = 0; fin_idx < FINGER_SIZE; fin_idx++) {
      fprintf(fp, "%3d %3d%c",
              straight_in[set_bright].finger[fin_idx],
              straight_out[set_bright].finger[fin_idx],
              (fin_idx == FINGER_SIZE-1 ? '¥n' : ' '));
    }
#endif
  }
  fclose(fp);

  /**  Create Bright Level Data File  **/
  bright_level_set();

  for(fp = file_name_create(BEND_FILE), set_bright = 0;
      set_bright < SET_BRIGHT_CNT; set_bright++) {
#ifdef  FLEX_ON
    fprintf(fp, "%3d ", set_bright);
    for(fin_idx = 0; fin_idx < FINGER_SIZE; fin_idx++) {
      fprintf(fp, "%3d %3d%c",
              bend_in[set_bright].finger[fin_idx],
              bend_out[set_bright].finger[fin_idx],
              (fin_idx == FINGER_SIZE-1 ? '¥n' : ' '));
    }
#endif
  }
  fclose(fp);
}


static  void  bright_level_set()
{
  FILE    *fp;
  register int  idx, fin_idx;
  int     set_bright, set_dim, select_bright, select_dim;
  int     invalid_data, sub_tot, sub_tot_save;
  short   firstp, invalidp, data_existp;

  for(invalid_data = INVALID_THRESH, data_existp = 0;
      !data_existp && invalid_data > 0; invalid_data--) {
    for(firstp = 1, set_bright = 0;
        set_bright < SET_BRIGHT_CNT; set_bright++) {
      for(invalidp = fin_idx = 0; fin_idx < FINGER_SIZE; fin_idx++) {
        if (((straight_in[set_bright].finger[fin_idx] -
              bend_in[set_bright].finger[fin_idx]) < invalid_data) ||
            ((straight_out[set_bright].finger[fin_idx] -
              bend_out[set_bright].finger[fin_idx]) < invalid_data)) {
          invalidp = 1;
          break;
        }
      }
      if (invalidp) continue;
      data_existp = 1;
      for(sub_tot = 0, fin_idx = 1; fin_idx < FINGER_SIZE; fin_idx++) {
        sub_tot += abs(straight_in[set_bright].finger[fin_idx-1] -
                       straight_in[set_bright].finger[fin_idx]) +
                         abs(bend_in[set_bright].finger[fin_idx-1] -
                             bend_in[set_bright].finger[fin_idx]);
        sub_tot += abs(straight_out[set_bright].finger[fin_idx-1] -
                       straight_out[set_bright].finger[fin_idx]) +
```

```
                        abs(bend_out[set_bright].finger[fin_idx-1] -
                            bend_out[set_bright].finger[fin_idx]);
        }
        if (firstp) {
          firstp = 0;
          sub_tot_save = sub_tot;
          select_bright = set_bright;
#ifdef  DIM_ON
          select_dim = set_bright;
#endif
        }
        else {
          if (sub_tot_save > sub_tot) {
            sub_tot_save = sub_tot;
#ifdef  DIM_ON
            select_dim = select_bright;
#endif
            select_bright = set_bright;
          }
        }
      }
    }

    if (!data_existp) {
      fprintf(stderr, "Error : Can't Select Bright Level¥n");
      exit();
    }

    fp = file_name_create(BRIGHT_FILE);
#ifdef  FLEX_ON
    fprintf(fp, "%d", select_bright);
#ifdef  DIM_ON
    fprintf(fp, " %d¥n", select_dim);
#else
    fprintf(fp, "¥n");
#endif
#endif
}


static  FILE  *file_name_create(fname)
     char    fname[];
{
    FILE    *fp;
    char    new_fname[FILENAME_SIZE];
    int     fnum;

    for(fnum = 0; fnum < MAXFILENUM; fnum++) {
      sprintf(new_fname, "%s%02d", fname, fnum);
      if ((fp = fopen(new_fname, "r")) == NULL) {
        if ((fp = fopen(new_fname, "w")) == NULL) {
          fprintf(stderr, "Error : Can't Open %s File¥n", new_fname);
          exit(-1);
        }
        return(fp);
      }
      else
        fclose(fp);
    }
}
```

```
/*****************************************************************************
********************************** alignment.c ********************************
*****                                                                      *****
*****          This program allows to fix the axis of the reference,       *****
*****    related to the source. It creates the file CALIB_FILE, which contains *****
*****    data for axis orientation, position of the hemisphere (space where the *****
*****    magnetic field exists),...                                        *****
*****                                                                      *****
*****************************************************************************
*****************************************************************************/

#include "tracker.h"

FILE    *fd, *fopen();

main(argc, argv)
     int     argc;
     char    **argv;
{
  int     n, m, loop, station, method;
  double  x0, x1, x2, y0, y1, y2, z0, z1, z2, wx, wy, wz;
#ifdef  USE_SYSTEM_V
  double  sqrt(double);
#else
  double  sqrt();
#endif

  if ((fd = fopen(CALIB_FILE, "w")) == NULL) {
    printf("Can't open file %s¥n", CALIB_FILE);
    exit(-1);
  }

  printf("¥n¥n3 SPACE Alignment Utility");

  while(1) {

    while(1) {
      printf("¥n¥n Enter Source No. for Alignment or 0 to end : ");
      scanf("%d", &station);
      while(getchar() != '¥n');
      if (station < 0 || station >= 3) {
        printf("Source No. must be 1 or 2¥n");
        continue;
      }
      else break;
    }

    if (station == 0) {
      fclose(fd);
      exit(0);
    }

    trinit();
    trareset(station);
    sleep(1);

    trinit();
    if (station == 1) trset(1); else trset(16);
    if (station == 1)
      trhemis(1, 0.0, 0.0, -1.0);
    else
      trhemis(5, 0.0, 0.0, -1.0);

    while(1) {
      printf("¥n Select Alignment Axes¥n");
      printf("¥n    1 ---- X-Y Axes.");
      printf("¥n    2 ---- X-Z Axes.");
      printf("¥n Enter your choice  :  ");
      scanf("%d", &method);
      while(getchar() != '¥n');
      if (method <= 0 || method >= 3) {
        printf(" Enter 1 or 2 to select alignment axes.¥n");
        continue;
      }
      else break;
    }

    printf("¥nStep1 --- Place Sensor no.1 to Origin and press <CR>.");
    while(getchar() != '¥n');
    trposin();
    n = trposrd(&x0, &y0, &z0);

    printf("¥nStep2 --- Place Sensor no.1 to X-axis and press <CR>.");
    while(getchar() != '¥n');
    trposin();
    n = trposrd(&x1, &y1, &z1);

    if (method == 1)
      printf("¥nStep3 --- Place Sensor no.1 to Y-axis and press <CR>.");
    else
      printf("¥nStep3 --- Place Sensor no.1 to Z-axis and press <CR>.");
    while(getchar() != '¥n');

    trposin();
    n = trposrd(&x2, &y2, &z2);

    if (method == 2) {
      wx= -(y1-y0)*(z2-z0)+(y2-y0)*(z1-z0);
      wy= -(z1-z0)*(x2-x0)+(z2-z0)*(x1-x0);
```

```
        wz= -(x1-x0)*(y2-y0)+(x2-x0)*(y1-y0);
        x2=wx * 50.0 / sqrt(wx*wx + wy*wy + wz*wz) + x0;
        y2=wy * 50.0 / sqrt(wx*wx + wy*wy + wz*wz) + y0;
        z2=wz * 50.0 / sqrt(wx*wx + wy*wy + wz*wz) + z0;
      }

      printf("¥nAlignmet data is:¥n");
      printf("A%1d,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf¥n",
             station, x0, y0, z0, x1, y1, z1, x2, y2, z2);
      fprintf(fd,
              "A%1d,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf,%4.2lf¥n",
              station, x0, y0, z0, x1, y1, z1, x2, y2, z2);
    }
  }
```

```
/*****************************************************************************
************************************ dessin.c ********************************
*****                                                                       *****
*****              Definition of some simple graphical functions.           *****
*****                                                                       *****
*****************************************************************************
*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <gl.h>


/** To initialize the screen : opens a half-screen window **/
init_screen()
{
  prefposition(0, XMAXSCREEN/2, 0, YMAXSCREEN/2);
  winopen("test");
  ortho2(0, XMAXSCREEN/2, 0, YMAXSCREEN/2);
  RGBmode();
  doublebuffer();
  gconfig();
}

/** The following procedures are to draw some figures in the (x,y) plan; but in the **/
/** main program they are called with y and -z due to the orientation of the refe-  **/
/** rence related to the 3Space Tracker source                                      **/

/** Draws a filled circle; (x,y) is the center and r is the radius **/
draw_circle(x, y, r)
float x, y, r;
{
  circf(x, y, r);
}

/** Draws a square from the center (x,y), the side s and the inclination ang, respect to x axis **/
draw_square(x, y, s, ang)
float x, y, s, ang;
{
  float v0[2], v1[2], v2[2], v3[2]; /* these are the 4 corners that will be calcula-
                                       ted from the center */

  v0[0] = x - s*sin(M_PI/4 - M_PI*ang/180)/sqrt(2);
  v0[1] = y - s*cos(M_PI/4 - M_PI*ang/180)/sqrt(2);
  v1[0] = v0[0] + s*cos(M_PI*ang/180);
  v1[1] = v0[1] + s*sin(M_PI*ang/180);
  v2[0] = v0[0] + s*cos(M_PI/4 + M_PI*ang/180)*sqrt(2);
  v2[1] = v0[1] + s*sin(M_PI/4 + M_PI*ang/180)*sqrt(2);
  v3[0] = v0[0] - s*sin(M_PI*ang/180);
  v3[1] = v0[1] + s*cos(M_PI*ang/180);

  bgnpolygon();
  v2f(v0);v2f(v1);v2f(v2);v2f(v3);
  endpolygon();                                 /* the square is filled */
}
```

```
/*****************************************************************************
*****************************************************************************
*****                          Makefile                                 *****
*****************************************************************************
*****************************************************************************/

INC = -I$(HOME)/include
MLIB = -lm
GLLIB = -lgl
SUNLIB = -lsun -lbsd
HEADER1 = dg2cmd.h dg2def.h dg2ext.h dg2str.h dg2use.h get_time.h
HEADER2 = tracker.h
HEADER3 = network.h

default : dg2_calibrate alignment hand

dg2_calibrate : dg2_operate.o dg2_calibrate.c
        cc -g3 dg2_operate.o dg2_calibrate.c $(INC) -o dg2_calibrate $(MLIB)

alignment : tracker.o alignment.c
        cc -g3 tracker.o alignment.c -o alignment $(MLIB)

hand : dg2_operate.o double_get.o tracker.o network.o dessin.o hand.c
        cc -g3 dg2_operate.o double_get.o tracker.o network.o dessin.o hand.c -o hand $(INC) $(MLIB)
 $(GLLIB) $(SUNLIB)

dg2_operate.o : dg2_operate.c $(HEADER1)
        cc -c -g3 dg2_operate.c $(INC)

double_get.o : double_get.c $(HEADER1) $(HEADER2)
        cc -c -g3 double_get.c $(INC)

tracker.o : tracker.c $(HEADER2)
        cc -c -g3 tracker.c

network.o : network.c $(HEADER3)
        cc -c -g3 network.c $(MLIB)

dessin.o : dessin.c
        cc -c -g3 dessin.c $(MLIB) $(GLLIB) $(SUNLIB)
```