TR-H-226

Model of Motion Detector

Martin DAVID, Frans VERSTRATEN and
Nicolas SCHWEIGHOFER (ERATO)

1997.7.17

ATR人間情報通信研究所
〒619-02　京都府相楽郡精華町光台2-2　TEL: 0774-95-1011

ATR Human Information Processing Research Laboratories
2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02, Japan
Telephone: +81-774-95-1011
Fax : +81-774-95-1008

MODEL OF MOTION DETECTOR

based on Alexader Grunewald's papers[1 & 2]

developed by

MARTIN DAVID

under the supervision of

DR. FRANS VERSTRATEN

with the collaboration of

DR. NICOLAS SCHWEIGHOFER

IN

ATR

INTRODUCTION

The purpose of this model is to reproduce human perception regarding sharpening of perception, integration of similar direction transparent motion, independent perception of distinct transparent motion and motion after effect.

The front-end of the system is a vector of 24 units, corresponding to 24 directional selective cells (every 15°). This layer correspond to motion detector cells.

The output of the system is another vector of 24 units corresponding to higher cognitive level cells, namely perception cells.

The implementation of such system with a neural network needs a lot of tuning. Therefore, the program, written in C++ to allow great execution speed, is called by a MATLAB script. MATLAB handle the display and every parameter can be changed easily.

For an explanation of the model, its equations and the link with psycho-physics results, please refer to Alexander Grunewald`s papers[1 & 2]. For further detail on the mathematical model and the link with biology, please refer to Grossberg`s paper[3].

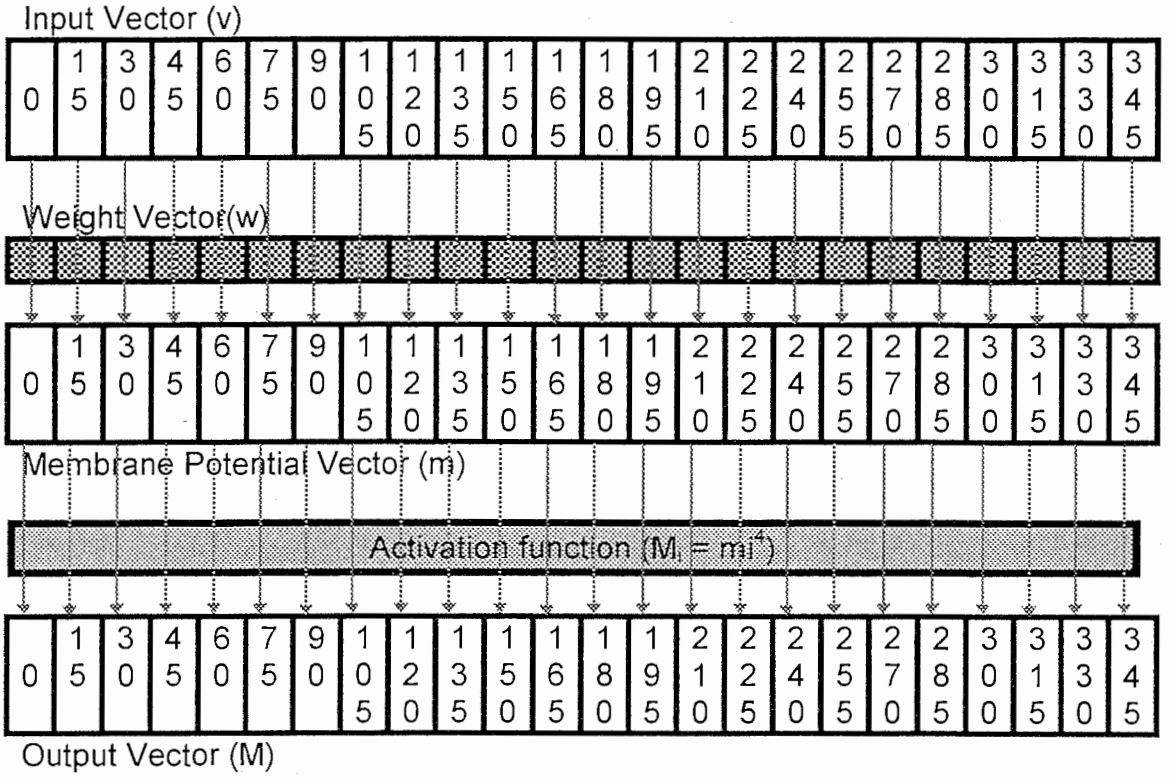
The current document shall serve as a user and programmer manual. It is divided in four sections:

1. Description of the model
2. Operator Manual
3. Programmer manual
    - C++
    - MATLAB

You will also find in annexe all the C++ code files and the matlab scripts.

1. DESCRIPTION OF THE MODEL

For in detail explanation bout this model please refer to [1,2&3].



Output Vector(m)

The network is define by two differential equations and an activation function:

$$\frac{dw_i}{dt} = R\left(1 - w_i\right) - v_i w_i$$

$$\frac{dm_i}{dt} = -m_i + \left(1 - m_i\right)\left(F_i^+ + B_i^+\right) - \left(1 + m_i\right)\left(F_i^- + B_i^-\right)$$

and

$$M_i = m_i^4$$

Where:

v_i is the i^{th} element of the input vector;
w_i is the i^{th} element of the weight vector;
m_i is the i^{th} element of the membrane potential vector;
M_i is the i^{th} element of the outputs vector;

R is a constant, in this case 0.5

F_i^+ is the convolution product between the Feedforward excitatory Kernels and the product element to element of v and w.
F_i^- is the convolution product between the Feedforward inhibitory Kernels and the product element to element of v and w.
B_i^+ is the convolution product between the Feedback excitatory Kernels and the output vector M.
B_i^- is the convolution product between the Feedback inhibitory Kernels and the output vector M.

2. USER MANUAL

2.1 Before starting

Everything is included in the compressed file:
/tmp_mnt/home/mac01/mdavid GRUN.tar.Z

The recover the files, you most:

- Uncompressed: `uncompress GRUN.tar.Z`
- Untar: `tar xvf GRUN`

All the files are divided in two directories: **src** and **MATLAB**.

The **src** directory contain the following files:

- `vector.C`
- `vector.h`
- `network.C`
- `network.h`
- `main.C`
- `default.h`
- `mask.h`
- `makefile`

To compile the process, you only have to type `make`. Since this process has been programmed in C++, you will need a C++ compiler. The default compiler is g++, but it could be change easily by editing the makefile. The option -O4 is used to obtain an optimized process but in case of problem, this option can be removed.

The **MATLAB** directory contain the following files:

- `init.m`
- `run.m`
- `stimuli.txt`

init.m and run.m are MATLAB scripts while stimuli.txt is the default stimuli file. This file should not be changed.

2.2 To start

- Start MATLAB by typing `MATLAB`
- Change to the proper directory
- Initialise parameter with `init`
- Execute a simulation by typing `run`
- Enlarge the figure window with the mouse

2.3 To change parameters

You can get a list of the parameters by typing the MATLAB command `who`.
You can get the value of any parameter by typing its name.
You can change the value of any parameter by typing its name, then `=` then the new value.
You can get the new simulation by typing `run` again.

2.4 Description of the parameters

2.4.1 Kernel related parameters

Among the four kernels three of them are define by gaussian curves and therefore can be describe by their gain and standard deviation. Note that the gaussian used here are not normalised. Therefore the gain is the value of the middle unit.
The last kernel is base on a sinusoide curve and only the gain can be changed.

Variable name	Description
CFM	Gain for Feedforward inhibition kernel used to compute F^-
CFP	Gain for Feedforward excitation kernel used to compute F^+
CBM	Gain for Feedback inhibition kernel used to compute B^-
CBP	Gain for Feedback excitation kernel used to compute B^+
SDFP	Standard deviation for Feedforward inhibition kernel
SDBP	Standard deviation for Feedback excitation kernel
SDBF	Standard deviation for Feedback excitation kernel

2.3.2 Parameter related to graphics

When the command run is type on the MATLA prompt, the script execute the UNIX process *mov* with the proper set of option. This process will produce two files: *left* and *right*. These are the files that will be displayed in the figure window. If you want to display a different vector, you have to change the string variables *LEFT* or *RIGHT* and re-run the simulation.

The possible options are:

Desired graph	Command to type
Inputs (v)	LEFT = '-INP '
Weights (w)	LEFT = '-w '
Product of $w_i * v_i$	LEFT = '-adapt '
Membrane potential (m)	LEFT = '-m '
Output (M)	LEFT = '-M '

Notes:

- The space at the end of the string is mandatory.
- Only the single back quote symbol is accepted by MATLAB.

Example:

```
>> LEFT = '-INP '
>> RIGHT = '-M '
>> run
```

This sequence will display the input layer to the left and the output layer to right. These are the default options.

The variable *LVIEW* and *RVIEW* control the point for each graph. You can also rotate the graphics using the mouse, but the point of view will be reset according to *LVIEW* and *RVIEW* each you will re-run the simulation.

2.3.3 Other parameters

UNIT1 and *UNIT2* control the stimuli. The input layer is numbered from 1 to 24, corresponding to angles numbered from 0 to 345 (note that $360^\circ = 0^\circ$).
One or two stimuli can be injected in the system, activation will be characterised by the designated unit at 9 and its immediate neighbours at 3. If the two signals are adjacent the activity will add up.
To keep only one stimulus, set the *UNIT2* at 0.

The *STRING* variable allow you had any option while executing the **mov** process.

2. PROGRAMMER`S GUIDE

2.1 MATLAB scripts

There are only two very simple MATLAB scripts: one to initialise variable, and the other to execute the simulation and display the graphics.

You will find the script in Annexe C.

2.1.1 init.m

This script initialise all MATLAB variable.

2.1.2 run.m

This script will call start the simulation according to the variable set in MATLAB.

Note that the process use number in the interval [0 23] to identify the units while MATLAB use [1 24].

2.2 C++ Program

2.2.1 Use guide of *mov*

Although it is a lot easier to call the **mov** process through the MATLAB script, this process is a UNIX executable that can be execute just by typing `mov` to the UNIX prompt.

All the option are describe through the option `-h`.

example:

if you type:

```
mov -h
```

you obtain:

```
        sytaxe: mov [options]
OPTIONS:
             -h                    help
        -stimul <filename>  name of the stimuli file
        -fgain_i <value>    gain of the forward inhibition
        -fstdev_i  <value>       standard  deviation  of  the  forward
inhibition
        -fgain_e <value>    gain of the forward excitation
        -fstdev_i  <value>       standard  deviation  of  the  forward
inhibition
        -bgain_i <value>    gain of the feedback inhibition
        -fstdev_i  <value>       standard  deviation  of  the  forward
inhibition
        -bgain_e <value>    gain of the feedback excitation
        -fstdev_i  <value>       standard  deviation  of  the  forward
inhibition
        -time <value>       duration of the total stimulation
        -dt   <value>       time step
        -interval<value>     time interval between 2 points written in
files
        -M <filename>          if present, will write output layer in
<filename>
        -INP <filename>    if present, will write input layer (stimuli)
in <filename>
        -adapt <filename>       if present, will write adapted layer in
<filename>
        -m <filename>       if present, will write m layer in<filename>
        -MAE                  if present, a motion after effect will be
simultated
-GROSS              if present, original grossberg's equation will be
used
```

2.2.2 Default values

The default value are written in the file *default.h*, but to make any change in this file effective, to process most be re-compile.
You will also find in *mask.h* the value taken from Grunewald`s paper's curve. Only the forward inhibition mask had been taken from curve.

2.2.3 The source code

The source code is distributed in three files: vector.C, network.C and main.C. The process had been implanted and C++ and two classes were created: vector and network.

a) The class vector

A vector is an array of double precision real number (in C, type *double*) with and integer to specify its length.

The destructors and contructors allow to allocate and deallocate memory at the execution time and therefore the vector can be of any length.

Further more, a few operators had been add like convolution, multiplication, activation, gaussian, addition.

Some I/O functions are also implanted allowing display on the screen (print) and disk access in ASCII(appendTXT, readTXT, writeTXT).

For better understanding see the code in annexe A

b) The class network

The idea to make a specific class for this network was to avoid complication inherent to global/local/static variable. Further more, the constructors and destructors will take care of the memory allocation.
The class contain aall information about the network namely: all the vectors and kurnels with their characteristics (standard deviation, gain).

The function *epoque* serve as a main function. One iteration of that function correspond to one time step. The differential equations are written in *deriv_w* and *deriv_m* functions. The method of integration used is Runge-Kutta (see *Runge_m* and *Runge_w*).

For better understanding, see code in annexe A

c) The functions *main* and *parseopt*

The file **main.C** contains the main function of the process. The parsing of the command line occurs in this function and the network in initialise according to the result of that parsing.
This also that function that take care of writing in files.
If other options have to be add to this process those should be control by the *main* function and properly explain in the *print_help* function.

You will also find this code in Annexe A

ANNEXE A: C++ SOURCE FILES

```
1     #define DT 0.01
2     #define TIME 3.0
3     #define INTERVAL 0.1
4     #define STIMULI "stimuli.txt"
5     #define FGAIN_I 12.0
6     #define FGAIN_E 18.0
7     #define BGAIN_I 600.0
8     #define BGAIN_E 410.0
9     #define FSTDEV_E 1.0
10    #define BSTDEV_I 6.0
11    #define BSTDEV_E 0.3
12
13
14
15
```

```
1
2       #define FB_MASK_I_L 25
3       double  FB_MASK_I[FB_MASK_I_L] = { 0.018, 0.023, 0.026, 0.031, 0.036, 0.043, 0.047, 0
        .054, 0.059, 0.062, 0.066, 0.067, 0.070, 0.067,0.066, 0.062, 0.059, 0.054, 0.047, 0.04
        3, 0.036, 0.031,0.026, 0.023, 0.018 };
4
5       #define FB_MASK_E_L 3
6       double FB_MASK_E[FB_MASK_E_L] = {0.0, 1.0, 0};
7
8       #define FF_MASK_I_L 25
9       double FF_MASK_I[FF_MASK_I_L] = {0.075, 0.070, 0.063, 0.054, 0.046, 0.037, 0.028, 0.0
        19, 0.017, 0.005, 0.001, 0.00, 0.0, 0.000, 0.001, 0.005, 0.017, 0.019, 0.028, 0.037, 0
        .046, 0.054, 0.063, 0.070, 0.075};
10
11      #define FF_MASK_E_L 11
12      double FF_MASK_E[FF_MASK_E_L] = {0.0, 0.001, 0.03, 0.12, 0.27, 0.39, 0.27, 0.12, 0.03
        , 0.001, 0.0};
13
14
15
```

```
1     #include "vector.h"
2
3
4     class network
5     {
6     private:
7
8     public:
9       vector INP;
10      vector w;
11      vector old_w;
12      vector Fatigue;
13      vector old_fatigue;
14      vector fb_mask_i;
15      vector fb_mask_e;
16      vector ff_mask_i;
17      vector ff_mask_e;
18      vector Fplus;
19      vector Fminus;
20      vector Bplus;
21      vector Bminus;
22      vector Total_i;
23      vector Total_e;
24      vector m;
25      vector old_m;
26      vector M;
27      double fgain_i,  fgain_e,  bgain_i,  bgain_e;
28      double           fstdev_e, bstdev_i, bstdev_e;
29      int verbose;
30      int GROSSBERG;
31
32      network();
33      network(int);
34      ~network();
35      setup(int);
36      void init_mask();
37      void init_mask_gauss();
38
39      void init();
40      void init(double*);
41      void init(const vector&);
42      vector& epoque(double);
43
44      void update_input();
45      void Runge_w(double);
46      void Euler_w(double);
47      void Runge_m(double);
48      void Euler_m(double);
49      void activation();
50      double deriv_w(double,double);
51      double deriv_m(double,double,double);
52    };
53
54
```

```
1     #include <stdio.h>
2     #include <stdlib.h>
3     #include <stream.h>
4     #include <fstream.h>
5     #include <math.h>
6
7     #define PI 3.1415
8
9     double valeur_absolue(double);
10
11    class vector
12    {
13    private:
14        double *mat;
15        int length;
16    public:
17    // Constructors and destructors
18      vector(int);
19      vector(char*);
20      vector();
21      setup(int);
22      ~vector();
23      destroy();
24
25    // accessor
26      double& operator[] (const int x)       {return(mat[x]);};
27      double operator[]  (const int x) const {return(mat[x]);};
28      int const getlength() const {return(length);};
29      double* getmat() const {return(mat);};
30
31    // operators
32      void operator=(double);
33      void operator=(vector);
34
35    // operations
36      void conv(const vector&,const vector&);
37      void operator=(double[]);
38      void mult(double);
39      int mult(const vector&,const vector&);
40      int add(const vector&,const vector&);
41
42    // Initialisation
43      int gaussian(double,double);
44      int sinusoide(double);
45      void activate(int);
46
47    // I/O
48      void print();
49      int appendTXT(char*);
50      int writeTXT(char*);
51      int readTXT(char*);
52    );
53
54
55
56    // non-member functions related to convolution
57    int wrap(int,int);
58    double masking(const vector&,const vector&,int);
```

```
1   #include "network.h"
2   #include "default.h"
3   #define version "Motion detector, based of Grunewald's equation. By Martin DAVID"
4
5   int parseopt(const char* name, char* val, int argc, char** argv)
6   {
7       for (int i=1;i<argc;i++)
8       {
9           if (!strcmp(name, argv[i]))
10          {
11              if (i<argc-1)   strcpy(val,argv[i+1]);
12              else val="";
13              return(1);
14          }
15      }
16      return(0);
17  }
18
19  void print_help(char* process_name)
20  {
21      cout << "Syntaxe: " << process_name << " [options]" << endl;
22      cout << "OPTIONS:" << endl;
23      cout << "       -h                  help" << endl;
24      cout << "       -stimul <filename>  name of the stumuli file" << endl;
25      cout << "       -fgain_i <value>    gain of the forward inhibition" << endl;
26      cout << "       -fstdev_i <value>   standart deviation of the forward inhibition"
     << endl;
27      cout << "       -fgain_e <value>    gain of the forward excitation" << endl;
28      cout << "       -fstdev_i <value>   standart deviation of the forward inhibition"
     << endl;
29      cout << "       -bgain_i <value>    gain of the feedback inhibition" << endl;
30      cout << "       -fstdev_i <value>   standart deviation of the forward inhibition"
     << endl;
31      cout << "       -bgain_e <value>    gain of the feedback excitation" << endl;
32      cout << "       -fstdev_i <value>   standart deviation of the forward inhibition"
     << endl;
33      cout << "       -time <value>       duration of the total stimulation" << endl;
34      cout << "       -dt   <value>       time step" <<endl;
35      cout << "       -interval<value>    time interval between 2 points written in fil
    es" << endl;
36      cout << "       -M <filename>       if present, will write output layer in <filen
    ame>" << endl;
37       cout << "       -INP <filename>    if present, will write input layer (stimuli)
    in <filename>" << endl;
38      cout << "       -adapt <filename>       if present, will write adapted layer in <
    filename>" << endl;
39      cout << "       -m <filename>       if present, will write m layer in <filename>"
     << endl;
40      cout << "       -MAE                if present, a motion after effect will be sim
    ultated" << endl;
41      cout << "       -GROSS              if present, original grossberg's
42  equation will be used" << endl;
43      exit(0);
44    }
45
46  void main(int argc, char **argv)
47  {
48    int nb_detectors = 24;
49    network motion(nb_detectors);
50    vector output_layer(nb_detectors);
51    char val[80];
52    vector stimuli(nb_detectors);
53    motion.init();
54
55
56  // ~~~~~~~~~~~~~~~~~~~PARSING OF THE COMMAND LINE, SeTTING DEFAULT~~~~~~~~~~
57    if (parseopt("-h", val, argc, argv))  print_help(argv[0]);
58    motion.verbose = (parseopt("-v", val, argc, argv));
59    if (parseopt("-VER", val, argc, argv)) {cout << version << endl;exit(0);}
60
61    int MAE = (parseopt("-MAE", val, argc, argv));
```



```
62
63    if (parseopt("-stimul", val, argc, argv))
64      motion.INP.readTXT(val);
65      else motion.INP.readTXT(STIMULI);
66
67    motion.fgain_i = (parseopt("-fgain_i", val, argc, argv))?atof(val):FGAIN_I;
68    motion.fgain_e = (parseopt("-fgain_e", val, argc, argv))?atof(val):FGAIN_E;
69    motion.bgain_i = (parseopt("-bgain_i", val, argc, argv))?atof(val):BGAIN_I;
70    motion.bgain_e = (parseopt("-bgain_e", val, argc, argv))?atof(val):BGAIN_E;
71    motion.fstdev_e= (parseopt("-fstdev_e", val, argc, argv))?atof(val):FSTDEV_E;
72    motion.bstdev_i= (parseopt("-bstdev_i", val, argc, argv))?atof(val):BSTDEV_I;
73    motion.bstdev_e= (parseopt("-bstdev_e", val, argc, argv))?atof(val):BSTDEV_E;
74
75    double dt = (parseopt("-dt", val, argc, argv))?atof(val):DT;
76    double time = (parseopt("-time", val, argc, argv))?atof(val):TIME;
77    double interval = (parseopt("-interval", val, argc, argv))?atof(val):INTERVAL;
78
79    ofstream Mfile,mfile,adaptfile,wfile,INPfile;
80    if (parseopt("-M", val, argc, argv)) Mfile.open(val);
81    if (parseopt("-m", val, argc, argv)) mfile.open(val);
82    if (parseopt("-adapt", val, argc, argv)) adaptfile.open(val);
83    if (parseopt("-w", val, argc, argv)) wfile.open(val);
84    if (parseopt("-INP", val, argc, argv)) INPfile.open(val);
85
86    if (parseopt("-unit1", val, argc, argv)) motion.INP.activate(atoi(val));
87    if (parseopt("-unit2", val, argc, argv))
88      if (atoi(val)!=-1) motion.INP.activate(atoi(val));
89    if (parseopt("-gauss", val, argc, argv)) motion.init_mask_gauss();
90    else motion.init_mask();
91    motion.GROSSBERG = (parseopt("-GROSS",val,argc,argv));
92    if(motion.verbose)
93    {
94      cout << "STIMULI" << endl;
95      motion.INP.print();
96      cout << "GAINS:" << endl;
97      cout << "       CF+:  " << motion.fgain_e << endl;
98      cout << "       CF-:  " << motion.fgain_i << endl;
99      cout << "       CB+:  " << motion.bgain_e << endl;
100     cout << "       CB-:  " << motion.bgain_i << endl;
101     cout << "       SDF+: " << motion.fstdev_e <<endl;
102     cout << "       SDB+: " << motion.bstdev_e <<endl;
103     cout << "       SDB-: " << motion.bstdev_i <<endl;
104   }
105
106 //~~~~~~~~~~~~SIMULATION WITH STIMULI~~~~~~~~~~~~~~~~~
107   for(double timer=0;timer<time;timer+=interval)
108   {
109     for(int i=0;i<nb_detectors;i++)
110     {
111       if (mfile) mfile << motion.m[i] << " ";
112       if (Mfile) Mfile << motion.M[i] << " ";
113       if (adaptfile) adaptfile << motion.Fatigue[i]   << " ";
114       if (wfile) wfile << motion.w[i] << " ";
115       if (INPfile) INPfile << motion.INP[i] << " ";
116     }
117     if (mfile) mfile << endl;
118     if (Mfile) Mfile << endl;;
119     if (adaptfile) adaptfile << endl;
120     if (wfile) wfile << endl;
121     if (INPfile) INPfile << endl;
122
123     for(float j=0; j<interval; j+=dt)
124     {
125       output_layer = motion.epoque(dt);
126       //    for(int i=0;i<25;i++)
127       // fout << i << " " << time << " " << motion.m[i] << endl;
128
129     }
130   }
131 //~~~~~~~~~~~Motion After Effect~~~~~~~~~~~~~~~~~~~~
132   if (MAE)
```

```
133     {
134       motion.INP =1;
135       for(float timer=time;timer<time+2;timer+=interval)
136       {
137         for(int i=0;i<nb_detectors;i++)
138         {
139           if (mfile) mfile << motion.m[i] << " ";
140           if (Mfile) Mfile << motion.M[i] << " ";
141           if (adaptfile) adaptfile << motion.Fatigue[i]   << " ";
142           if (wfile) wfile << motion.w[i] << " ";
143           if (INPfile) INPfile << motion.INP[i] << " ";
144         }
145         if (mfile) mfile << endl;
146         if (Mfile) Mfile << endl;;
147         if (adaptfile) adaptfile << endl;
148         if (wfile) wfile << endl;
149         if (INPfile) INPfile << endl;
150         for(float j=0; j<interval; j+=dt)
151         {
152           output_layer = motion.epoque(dt);
153         //     for(int i=0;i<25;i++)
154         // fout << i << " " << time << " " << motion.m[i] << endl;
155         }
156       }
157     }
158     if (mfile) mfile.close();
159     if (Mfile) Mfile.close();
160     if (adaptfile) adaptfile.close();
161     if (wfile) wfile.close();
162     if(INPfile) INPfile.close();
163   }
```

```
1     #include "network.h"
2     #include "mask.h"
3
4
5     network::~network()
6     {
7     }
8
9     network::network()
10    {
11      setup(0);
12    }
13
14    network::network(int x)
15    {
16      setup(x);
17     }
18
19    network::setup(int x)
20    {
21      ff_mask_i.setup(FF_MASK_I_L);
22      ff_mask_e.setup(FF_MASK_E_L);
23      fb_mask_i.setup(FB_MASK_I_L);
24      fb_mask_e.setup(FB_MASK_E_L);
25
26      INP.setup(x);
27      w.setup(x);
28      Fatigue.setup(x);
29
30      Fplus.setup(x);
31      Fminus.setup(x);
32      Bplus.setup(x);
33      Bminus.setup(x);
34
35      Total_i.setup(x);
36      Total_e.setup(x);
37
38      m.setup(x);
39      M.setup(x);
40    }
41
42    void network::init_mask()
43    {
44      ff_mask_i = FF_MASK_I;
45      ff_mask_i.mult(fgain_i);
46      ff_mask_i[0]=0;
47    //  ff_mask_i.sinusoide(1);
48      if (verbose)
49      {
50        cout << "Forward inhibition mask:";
51        ff_mask_i.print();
52      }
53
54
55    //  ff_mask_e.gaussian(fstdev_e,fgain_e);
56      ff_mask_e = FF_MASK_E;
57      ff_mask_e.mult(fgain_e);
58      if (verbose)
59      {
60        cout << "Forward excitation mask:";
61        ff_mask_e.print();
62      }
63
64    //  fb_mask_i.gaussian(bstdev_i,bgain_i);
65      fb_mask_i = FB_MASK_I;
66      fb_mask_i.mult(bgain_i);
67      fb_mask_i[0]=0;
68      if (verbose)
69      {
70        cout << "Backward inhibition mask:";
71        fb_mask_i.print();
```



```
72      }
73
74
75    //  fb_mask_e.gaussian(bstdev_e,bgain_e);
76      fb_mask_e = FB_MASK_E;
77      fb_mask_e.mult(bgain_e);
78      if (verbose)
79      {
80        cout << "Backward excitation mask";
81        fb_mask_e.print();
82      }
83    }
84
85    void network::init_mask_gauss()
86    {
87      ff_mask_i = FF_MASK_I;
88      ff_mask_i.mult(fgain_i);
89      ff_mask_i[0]=0;
90    //  ff_mask_i.sinusoide(1);
91      if (verbose)
92      {
93        cout << "Forward inhibition mask:";
94        ff_mask_i.print();
95      }
96
97
98      ff_mask_e.gaussian(fstdev_e,fgain_e);
99      if (verbose)
100     {
101       cout << "Forward excitation mask:";
102       ff_mask_e.print();
103     }
104
105     fb_mask_i.gaussian(bstdev_i,bgain_i);
106     fb_mask_i[0]=0;
107   //  fb_mask_i[fb_mask_i.getlength()/2]=0;
108     if (verbose)
109     {
110       cout << "Backward inhibition mask:";
111       fb_mask_i.print();
112     }
113
114
115     fb_mask_e.gaussian(bstdev_e,bgain_e);
116     if (verbose)
117     {
118       cout << "Backward excitation mask";
119       fb_mask_e.print();
120     }
121   }
122
123
124   void network::init()
125   {
126     w = 1.0;
127     m =0.0;
128     M = 0.0;
129     Fatigue = 0.0;
130   }
131
132   void network::init(const vector& input_vect)
133   {
134     INP = input_vect;
135      w = 1.0;
136     m =0.0;
137     M = 0.0;
138     Fatigue = 0.0;
139   }
140
141   void network::update_input()
142   {
```

```
143   }
144
145   vector&  network::epoque(double dt)
146   {
147     update_input();
148     Fatigue.mult(INP,w);
149     Runge_w(dt);
150
151     Fminus.conv(Fatigue,ff_mask_i);
152     Fplus.conv( Fatigue,ff_mask_e);
153     Bminus.conv(M,fb_mask_i);
154     Bplus.conv (M,fb_mask_e);
155     Total_i.add(Fminus,Bminus);
156     Total_e.add(Fplus ,Bplus);
157     Runge_m(dt);
158     activation();
159     return m;
160
161   }
162
163   // Differential equations solving functions
164   double network::deriv_w(double w, double Fatigue)
165   {
166   //  cout << (0.5*(1-w)-Fatigue) << "  ";
167     return(0.5*(1-w)-Fatigue);
168   }
169
170   void network::Euler_w(double dt)
171   {
172     for (int i=0;i<INP.getlength();i++)
173        w[i]+=deriv_w(w[i],Fatigue[i])*dt;
174   //  cout << endl;
175   }
176
177
178   void network::Runge_w(double dt)
179   {
180     double k1,k2,k3,k4;
181     for(int i=0;i<INP.getlength();i++)
182     {
183       k1 = dt * deriv_w(w[i],Fatigue[i]);
184       k2 = dt * deriv_w(w[i]+k1/2.0,Fatigue[i]);
185       k3 = dt * deriv_w(w[i]+k2/2.0,Fatigue[i]);
186       k4 = dt * deriv_w(w[i]+k3,Fatigue[i]);
187       w[i] += ((k1+2*(k2+k3)+k4)/6.0);
188     }
189   }
190
191   double network::deriv_m(double m, double Total_i, double Total_e)
192   {
193   //  cout << (-1*m + (1-m)*Total_e - (1+m)*Total_i);
194     if (GROSSBERG) return(-1*m + (1-m)*Total_e - (m)*Total_i);
195     else return(-1*m + (1-m)*Total_e - (1+m)*Total_i);
196   }
197
198
199   void network::Euler_m(double dt)
200   {
201     for (int i=0;i<INP.getlength();i++)
202        m[i]+=deriv_m(m[i],Total_i[i],Total_e[i])*dt;
203   //  cout << endl;
204   }
205   void network::Runge_m(double dt)
206   {
207     double k1,k2,k3,k4;
208     for(int i=0;i<INP.getlength();i++)
209     {
210       k1 = dt * deriv_m(m[i],Total_i[i],Total_e[i]);
211       k2 = dt * deriv_m(m[i]+k1/2.0,Total_i[i],Total_e[i]);
212       k3 = dt * deriv_m(m[i]+k2/2.0,Total_i[i],Total_e[i]);
213       k4 = dt * deriv_m(m[i]+k3,Total_i[i],Total_e[i]);
```



```
214       m[i] += ((k1+2*(k2+k3)+k4)/6.0);
215     }
216   }
217
218   void network::activation()
219   {
220     for(int i=0;i<m.getlength();i++)
221       M[i] = (m[i]<=0)?0:m[i]*m[i]*m[i]*m[i];
222   }
223
224   // I/O function
225
226
227
228
229
230
231
232
233
234
```

```
1     #include "vector.h"
2
3
4     vector::vector()
5     {
6       length=0;
7       mat = NULL;
8     }
9
10    vector::vector(char *filename)
11    {
12      readTXT(filename);
13    }
14
15
16    vector::setup(int x)
17    {
18      length=x;
19      mat = new double[x];
20    }
21
22    vector::vector(int x)
23    {
24      setup(x);
25    }
26
27    vector::destroy()
28    {
29      length=0;
30      delete[] mat;
31    }
32
33    vector::~vector()
34    {
35      destroy();
36    }
37
38    // Operators
39    void vector::operator=(double x)
40    {
41      for(int i=0;i<length;i++)
42        mat[i]=x;
43    }
44
45
46    void vector::operator=(vector a)
47    {
48      if(length!=a.getlength()) cout << "Vector of different format, cannot
49    perform = opration";
50      else
51        for(int i=0;i<length;i++)
52          mat[i]=a[i];
53    }
54
55    void vector::operator=(double *array)
56    {
57      for(int i=0;i<length;i++)
58        mat[i]=array[i];
59    }
60
61
62    void vector::print()
63    {
64      for(int i=0;i<length;i++)
65        cout << mat[i] << "  ";
66      cout << endl;
67    }
68
69
70    // Operation
71    void vector::conv(const vector& a, const vector& mask)
```



```
72    {
73      if (mask.getlength()%2!=1)
74        {
75          cout << "Cannot perform convolution: mask dimension is even("
76               << mask.getlength() << ")" << endl;
77          exit(0);
78        }
79    //  if (mask.getlength()>a.getlength())
80    //    {       cout << "Cannot perform convolution: mask is bigger than input vector" <
      < endl;
81    //          exit(0);
82    //    }
83      if  (a.getlength()!=length)
84        {       cout << "Cannot perform convolution: input and out vector have different d
      imensions" << endl;
85              exit(0);
86        }
87      for(int i=0;i<length;i++)
88      {
89        mat[i]=masking(a,mask,i);
90      }
91    }
92
93
94    int wrap(int number,int length)
95    {
96      if (number>=length) return(number-length);
97      else if(number<0) return(number+length);
98           else return number;
99    }
100
101   double masking(const vector& a, const vector& mask, int pos)
102   {
103     double sum=0;
104     int j=wrap(pos-mask.getlength()/2,a.getlength());
105
106     for(int i=0;i<mask.getlength();i++)
107       sum+=a[wrap(j+i,a.getlength())]*mask[i];
108     return sum;
109   }
110
111   void vector::mult(double A)
112   {
113     for(int i=0;i<length;i++)
114       mat[i]*=A;
115   }
116
117   vector::mult(const vector& a, const vector& b)
118   {
119     if ((a.getlength()!=b.getlength())||(a.getlength()!=length))
120     {
121       cout << "Impossible to perform multiplication: vectors of different dimension"
122            << endl;
123       return 0;
124     }
125     for(int i=0;i<length;i++)
126        mat[i]=a[i]*b[i];
127        return 1;
128   }
129
130   vector::add(const vector& a, const vector& b)
131   {
132     if ((a.getlength()!=b.getlength())||(a.getlength()!=length))
133     {
134       cout << "Impossible to perform addition: vectors of differentdimension"
135            << endl;
136       return 0;
137     }
138     for(int i=0;i<length;i++)
139        mat[i]=a[i]+b[i];
140        return 1;
```

```
141   }
142
143   // Initialisation
144
145   double valeur_absolue(double x)
146   {
147     if (x<0) return -1*x;
148     else return x;
149   }
150
151   int vector::gaussian(double stdev,double gain)
152   {
153     if (!length%2) {cout<<"The vector needs to have an Odd length" <<endl;return(0);}
154     for(int i=0;i<length;i++)
155     {
156   //    mat[i]= gain*(1/(stdev*sqrt(2*PI))) * exp(-0.5*pow((i-length/2)/stdev,2));
157         mat[i]= gain* exp(-0.5*pow((i-length/2)/stdev,2));
158
159     }
160     return 1;
161   }
162
163   int vector::sinusoide(double gain)
164   {
165     if (!length%2) {cout<<"The vector needs to have an Odd length" <<endl;return(0);}
166     for(double i=0;i<length;i++)
167     {
168       mat[(int)i]=gain*(1-valeur_absolue(cos(PI*(i-length/2-1)/length)));
169     }
170     return 1;
171   }
172
173   void vector::activate(int i)
174   {
175     mat[wrap(i+1,length)]+=3;
176     mat[i]+=9;
177     mat[wrap(i-1,length)]+=3;
178   }
179
180   // I/O functions
181   int vector::writeTXT(char *filename)
182   {
183       ofstream fout;
184       fout.open(filename);
185       if (!fout)
186       {
187         cout<< "Impossible to open " << filename << " for writing" << endl;
188         return(0);
189       }
190       fout << length << endl;
191       for (int i=0;i<length;i++)
192       {
193         fout << mat[i] << " ";
194       }
195       fout.close();
196       return(1);
197   }
198
199   int vector::appendTXT(char *filename)
200   {
201       ofstream fout;
202       fout.open(filename,ios::app );
203       if (!fout)
204       {
205         cout<< "Impossible to open " << filename << " for writing"<< endl;
206         return 0;
207       }
208       fout << length << endl;
209       for (int i=0;i<length;i++)
210       {
211         fout << mat[i] << " ";
```



```
212       }
213       fout.close();
214       return(1);
215   }
216
217
218
219   int vector::readTXT(char *filename)
220   {
221       ifstream fin(filename);
222       if (!fin)
223       {
224         cout << "Impossible to open " << filename << " for reading." << endl;
225         return(0);
226       }
227       int l;
228       fin >> l;
229       if (l!=length)
230       {
231         destroy();
232         setup(l);
233       }
234       for (int i=0;i<length;i++)
235       {
236         fin >> mat[i];
237       }
238       fin.close();
239       return(1);
240   }
241
242
243
```

ANNEXE B MAKEFILE

```
1     FLAG = -O4
2     LIB = -lm
3
4     mov: vector.o network.o main.C
5             g++  ${FLAG} ${LIB} vector.o network.o main.C -o mov
6
7     vector.o: vector.C vector.h
8             g++ -c ${FLAG} vector.C -o vector.o
9
10    network.o: vector.o network.h network.C
11            g++ -c ${FLAG}  network.C -o network.o
12
13
14
15
```

ANNEXE C: MATLAB SCRIPTS

```
1       CFM = 12
2       CFP = 18
3       CBM = 600
4       CBP = 300
5
6       SDFP = 1
7       SDBP = 0.3
8       SDBM  = 6.0
9
10      UNIT1 = 10
11      UNIT2 = 16
12
13      PROCESS = 'mov '
14
15
16      LEFT = '-INP'
17      RIGHT = '-M'
18
19      STRING = ''
20
21      LVIEW = [-1 -8 8]
22      RVIEW = [-5 1 9]
23
```

```
1
2     unix([PROCESS ' ' LEFT ' left ' RIGHT ' right ' ' -fgain_i ' num2str(CFM)  ' -fgain_e
       ' num2str(CFP)  ' -bgain_i ' num2str(CBM)  ' -bgain_e ' num2str(CBP) ' -fstdev_e ' nu
      m2str(SDFP) ' -bstdev_e ' num2str(SDBP) ' -bstdev_i ' num2str(SDBM)  ' -MAE ' ' -unit1
       ' num2str(UNIT1-1) ' -unit2 ' num2str(UNIT2-1) STRING ' -gauss'])
3     disp([PROCESS ' ' LEFT ' left ' RIGHT ' right ' ' -fgain_i ' num2str(CFM)  ' -fgain_e
       ' num2str(CFP)  ' -bgain_i ' num2str(CBM)  ' -bgain_e ' num2str(CBP) ' -fstdev_e ' nu
      m2str(SDFP) ' -bstdev_e ' num2str(SDBP) ' -bstdev_i ' num2str(SDBM)  ' -MAE ' ' -unit1
       ' num2str(UNIT1-1) ' -unit2 ' num2str(UNIT2-1) STRING ' -gauss'])
4
5
6     load left
7     load right
8     subplot(121), surf(left)
9     title(['Left graph: ' num2str(LEFT) '                          Right graph: ' num2str(R
      IGHT)])
10    subplot(122), surf(right)
11    colormap(hsv)
12    subplot(121),rotate3d on
13    subplot(122),rotate3d on
14    subplot(121),view(LVIEW)
15    subplot(122),view(RVIEW)
16
17    title(['CFM:' num2str(CFM) '    CFP:' num2str(CFP) '    CBM:' num2str(CBM)      '
      CBP:' num2str(CBP)  '   STIMULI: ' num2str(UNIT1) ' & ' num2str(UNIT2)])
18
19
20
21
22
23
24
25
26
27
28
29
```