NeuralSyns Tutorial

Mafalda Sousa[1,2] and Paulo Aguiar[2,3]

1 - Faculdade de Medicina da Universidade do Porto
2 - Instituto de Biologia Molecular e Celular
3 - Centro de Matemática da Universidade do Porto

Janeiro 2013

Installing NeuralSyns

STEP 1

First you need the following libraries to be installed in your gnu-linux system (you may have some, if not all, already installed):

- Gsl (libgsl0-dev) : GNU Scientific Library, GSL
- Freeglut (freeglut3-dev) : an opensource alternative to the OpenGL Utility Toolkit
- WxWidgets (libwxbase2.8-dev, libwxgtk2.8-dev) : wxWidgets cross-platform C++ GUI toolkit (GTK+ development)
- Gtk (libgtk2.0-dev, libgtk2.0-common) : toolkit for creating graphical user interfaces
- Python (python-all-dev - python3-all-dev, libpython3.)

If you intend to use matplotlib together with python you also need:

- *python-matplotlib* : python based plotting system in a style similar to Matlab
- *python-gtk2-dev* : python GTK+ bindings, development files

Please notice that all these are well known, standard libraries and many linux distributions include them in their repositories, making the installation procedure totally straightforward. On Ubuntu you may simply use Synaptic and select the above mentioned packages.

STEP 2

Now that you have all the required libraries installed, you are ready to compile NeuralSyns in your system. Download the latest version of NeuralSyns from: *http://sourceforge.net/projects/NeuralSyns*. Extract the contents of the zip file to your home directory ($\sim$ */NeuralSyns*). To install NeuralSyns go to $\sim$ */NeuralSyns* and run the shell script "*NeuralSyns_install*":

```
$ unzip NeuralSyns.zip -d ~/
$ cd ~/NeuralSyns
$ chmod +x NeuralSyns_install
$ ./NeuralSyns_install
```

Running the "*NeuralSyns_install*" script will compile and link all source files and place three binaries in your $\sim$ */NeuralSyns/bin* directory: NeuralSyns, NetBuilder and ElectrodeBuilder. If you have correctly installed all the required libraries, you should have no errors, and the compilation takes just a few seconds. If you double checked everything (especially you confirmed that all the required libraries have been installed!) and you still are not able to compile NeuralSyns, send an email to: *neuralsyns@gmail.com*.

STEP 3

Next include $\sim$ */NeuralSyns/bin* in your PATH environment variable. Append the following line to your $\sim$ */.bashrc*, $\sim$ */.bash_profile* or $\sim$ */.profile* file (depends on your distribution setup; on Ubuntu append the $\sim$ */.bashrc* file):

```
export PATH=$PATH:~/NeuralSyns/bin
```

STEP 4

For NeuralSyns documentation go to $\sim$ */NeuralSyns/doc*. You can generate doxygen documentation regarding NeuralSyns code: simply type "doxygen" while inside $\sim$ */NeuralSyns/doc*; an html and a latex version will be created in different directories. A great deal of care has been taken in commenting the source code. We therefore expect the doxygen output to be useful.

Contents

1 Building a model of large populations neurons

An important advantage of NeuralSyns over other simulation tools is the small amount of effort required for building large network models with several different populations. In this tutorial we will have a glimpse of these capabilities by constructing a model for a feed-forward network with three populations types.
Apart from the neuron type of model that will be used, the following steps can be taken:

1. run NetBuilder and create the model;
2. run ElectrodeBuilder and define the stimuli protocols;
3. simulate the model with NeuralSyns;
4. analyze the results using DataAnalysis tools;

1.1 Motivation

As motivation consider the following neurobiological problem. Neural activity, in the form of excitatory signals, is transferred from population to population in neural networks. In this propagation process, it is crucial to keep the overall excitation levels bounded in order to maintain the system within appropriate functional parameters (which include proper signal-to-noise ratios). Neural systems use several strategies to produce this control but the most notable is through specialized interneuron populations. In order to produce a specific activity level in a target population in response to activity in a source population, parameters such as connectivity and synaptic efficacies must be within appropriate bounds. In addition, abnormally high excitation levels in the source population should be prevented from being transferred to the target population. In this tutorial, we will build and simulate a feed-forward network model which is capable of helping us to understand a bit better some of the key parameters controlling this activity transfer.

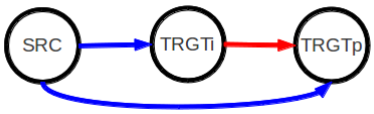

Figure 1: Network model architecture.

1.2 Model construction

Open a shell, change to your working directory and run the binary NetBuilder:

```
>> NetBuilder
```

This will open the NetBuilder graphical user interface (GUI). The NetBuilder GUI starts by presenting two options: loading a previously stored network model file or start a new model from scratch. Since we will build our first model from scratch, select the radio button *Create a network from scratch*. By pressing this option you activate an input box where you can enter the number of populations in your model. All models in NeuralSyns are composed by populations which aggregate neuron units with the same dynamical properties. For our model, type "3" and press the "Return" key (Fig. 2). Now we will define the basic properties of our populations.

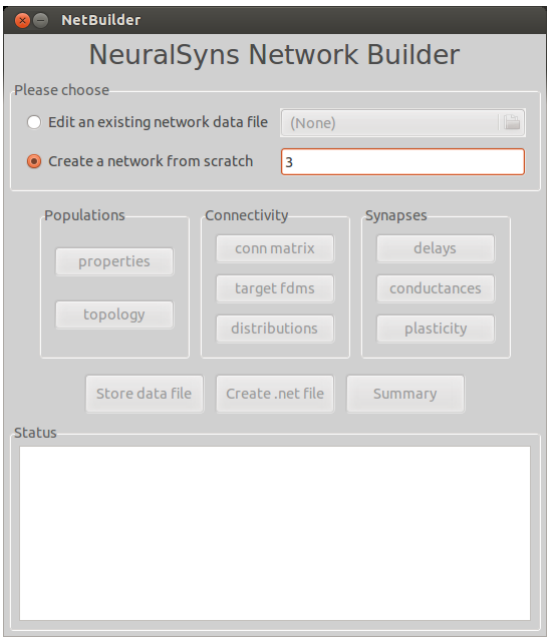

Figure 2: NetBuilder main frame: insert the number of populations.

1.2.1 Populations

Press the *properties* button. The *properties* button opens a new frame that allows us to set several population parameters. The *label* lets you define the populations' names. Please note that population labels must contain 10 or less characters. Call population 0 "SRC" (source), population 1 "TRGTp" (target, principal) and population 2 "TRGTi" target, interneurons) like in Fig. 3. Note that, population and neuron counting start at zero.

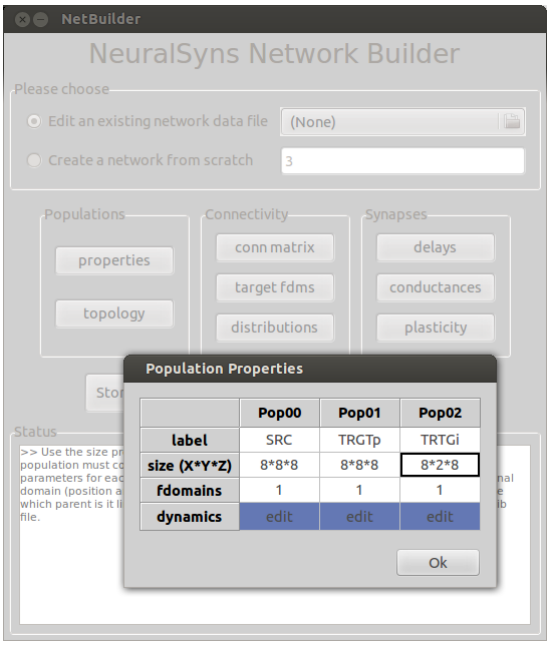

Figure 3: NetBuilder properties frame: define population properties.

The *size* of the population corresponds to the number of neuron units present in the population in a 3D space, and is defined in an array form. The effective placement of the population units in 3D space does not need to be constrained to a rectangular prism (in more detail in the *distribution button* description). In the "size" entry put "8*8*8", "8*8*8" and "8*2*8" for, respectively, population SRC, TRGTp and TRGTi.

NeuralSyns allows the creation of a small number of compartments attached to the soma compartment (which is the basic compartment). All these compartments are called in NeuralSyns *functional domains* and all neurons have at least one functional domain: the soma. The computational implications of these functional domains, or simply **fdomains**, are vast. A simple example is the segregation of inputs coming from different populations. In this tutorial we will create a population with two functional domains receiving excitatory and inhibitory connections, respectively. The goal is to demonstrate how to set up fdomains in NeuralSyns and produce a model which can be used to explore the dependence of synaptic efficacy with the synapse locus. Notice however that, by construction, NeuralSyns is not appropriate to solve in detail questions related with dendritic non-linear interactions, dendritic filtering, as well as all other space related modulations. These kind of questions should be analyzed in tools such as NEURON, which have the appropriate algorithms to simulate multi-compartmental models. The existence of fdomains in NeuralSyns serves only the purpose of approximating the effects of separated dendritic branches.
The neuronal response will be assessed in two situations: 1) excitatory and inhibitory connections target different dendritic fdomains, and 2) the excitatory connection targets the dendritic fdomain while the inhibitory connection targets the soma fdomain. The results will be discussed in further section of this tutorial.
In our model, type " 1", "2" and "1" in the *fdomains* entry type for population SRC, TRGTp and TRGTi, respectively. The *dynamics* entry allows editing the functional dynamic models and properties for each fdomain (Fig. 4).

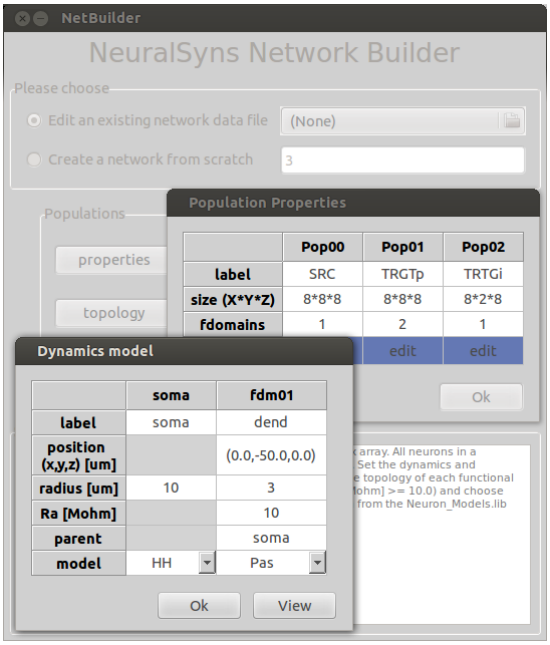

Figure 4: NetBuilder dynamics frame: define neuron models for each population

For populations SRC and TRGTi leave all the entries in the fdomains dynamics with their default values. Open the dynamic model frame of population TRGTp by selecting *edit*. For each fdomain you can define a fdomain *label*, which can be changed to "soma" and "dend", respectively. For the soma compartment you can define the *radius* length equal to 10, considering that the soma is a sphere. Notice that this geometry parameters are only for visualization propose.

Different dynamics can be assigned to different functional domains. NeuralSyns includes some neuronal models by default and more can be added (described in section 5). The included models are:

- HH, a modified Hodgkin-Huxley model
- LIF, a standard leaky integrate-and-fire model
- Izhikevich

- Passive, a standard passive-only model

The *model* entry allows the selection of the dynamical model by specifying the respective model parameters. For our model select "LIF" and "Pas" model for "soma" and "dend", respectively. For the integrate-and-fire parameters define: the C_m that sets the neuron membrane capacitance in nF; The R_m entry specifies the membrane resistance in Mohms; the V_{rest} sets the resting potential in mV, for the neuron units belonging to this population. The V_{thresh} sets the threshold voltage also in mV; Finally, use the $refrac\_period$ entry to define the absolute refractory period, in ms, for all neurons in this population. Leave all the parameters with their default values (Fig. 5)

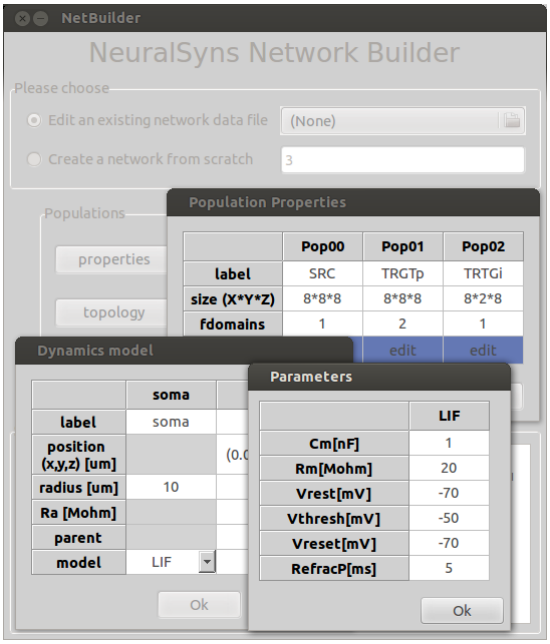

Figure 5: NetBuilder neuron model parameters

The fdomain "dend" is a compartment that is connect to his parent, defined in the entry *parent*. By default, all the compartments are connected to soma. However, they can be connected to other existing fdomains by changing the parent entry with the same label as the label defined in the parent fdomain.
The *position* entry is the absolute fdomain position in space. By default the soma position is $(0, 0, 0)$ and the other fdomains have a coordinate (x, y, z) relative with the soma position. The geometry of the compartments was defined as cylinders; the dimensions of the cylinder can be defined in the *radius* and by the *position* entry.
Finally, the neuron compartments structure can be visualized in the *View* button of the dynamics model frame, as illustrated in Fig. 6)
All the population properties are defined after pressing the *Ok* buttons in the opened windows and returning to the main GUI frame.

If you noticed, in the Main frame there's a *Status* textbox that is returning helpful information about each button that is pressed and several warning messages that allows seeing error choices before creating the final network file.

Now, we can define some parameters controlling the neurons placement in space, for each population. By pressing *topology* button, a topology frame is opened. For each population you can define an origin (X_0, Y_0, Z_0) of the box array in each direction of space. The spacing between neurons in each population is set by d_{mean} and d_{std}. These parameters define, respectively, the average distance and the standard deviation. Set our topology parameters as illustrated in fig. 7:

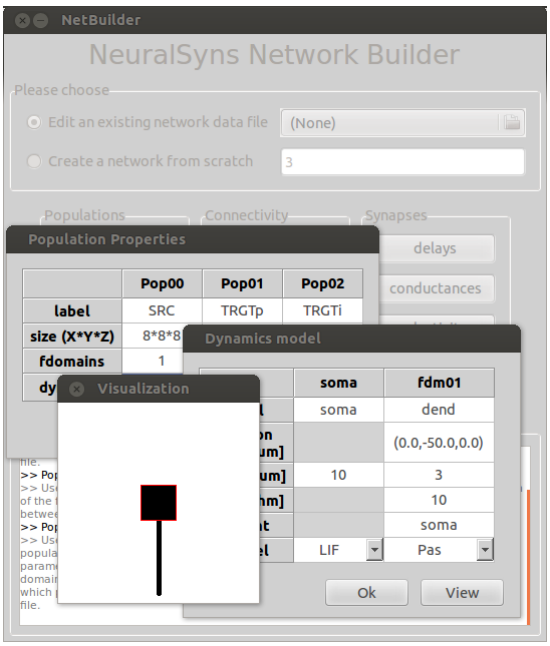

Figure 6: NetBuilder neuron compartments visualization.

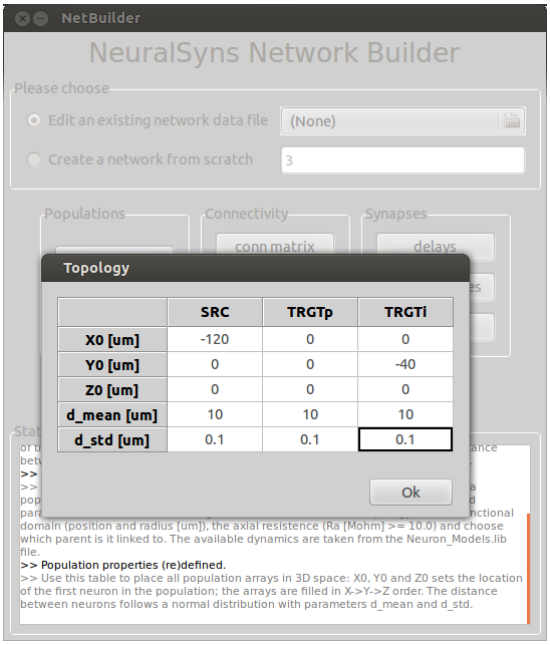

Figure 7: NetBuilder population topology.

1.2.2 Connectivity

Connectivity settings in NetBuilder are constructed on top of connectivity matrices, where lines set source populations and columns set target populations. These matrices provide a concise description of the connections present in the network model. The matrices do not state explicitly how each individual connection is set; what they provide is a statistical description of the connectivity.
We start by defining the parameters in the *Connectivity matrix* frame. Press the *conn matrix* button and set the following properties: each SRC neuron receives no inputs; each TRGTp neuron receives 40 synapses from the SRC population and 20 synapses from the TRGTi population; finally, each TRGTi neuron receives 40 synapses from the SRC population. The connectivity table should look like fig. 8. By pressing *Options* button in the connectivity matrix frame you have access to four connectivity properties:

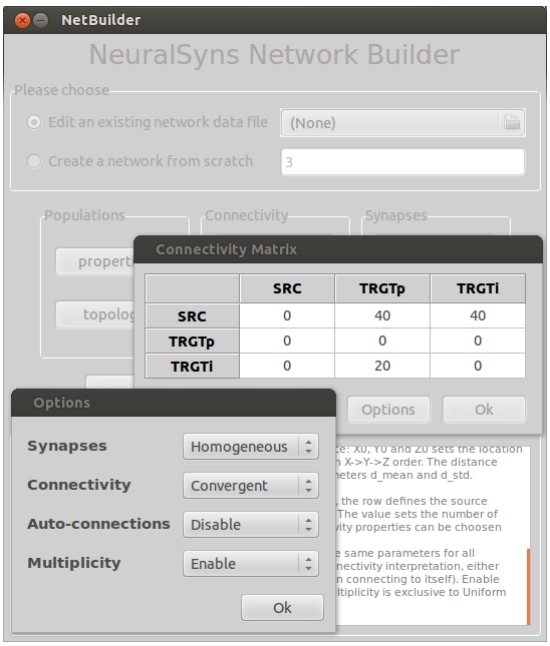

Figure 8: NetBuilder connectivity matrix: establish the connections between populations.

- Synapses can be Homogeneous/Heterogeneous;
- Connectivity can be Convergent/Divergent;
- Auto-connections can be Enable/Disable;
- Multiplicity can be Enable/Disable;

For our model, the Synapses are *homogeneous*, which means that the parameters for each connection type (matrix entry) are identical. This leads to a reduction in memory required by NeuralSyns to simulate the model. The connectivity table can either be a *convergent* connectivity table or a *divergent* connectivity table:

- **convergent**: each entry contains the number of inputs that each neuron, from the target population, receives from the source population; each input corresponds to a different source neuron.
- **divergent**: each entry contains the number of outputs that each neurons, from the source population, sends to the target population; each output corresponds to a different target neuron.

In this particular model we will adopt a convergent connectivity interpretation. Since we are not interest in modulating self-connections inside the same population, leave *Auto-connections* option "Disable". Enable *Multiplicity* option if you want to allow multiple connections from/to the same neuron. This option is exclusive for Uniform distribution. For our model you must "Disable" this option. Press *Ok* button to return to main frame.

Now by pressing *target fdms* button you can select the target compartment for each population. By default all the synapses are axo-somatic, but in our model we defined that population "SRC" is connected to dendritic compartment of "TRGTp". In the respective table entry, choose "dend" for the respective target fdomain of this pair of populations (SRC-TRTGp). Leave the other connections with their default values (see Fig. 9).

NeuralSyns also allows the user to define 3D receptive fields for each type of connections. The receptive fields are defined using Normal distributions with three independent standard deviations - σ_x, σ_y, σ_z. Sampling in the directions of space where the standard deviation is of the same order as the neuronal population dimension degenerates into a uniform sampling. By pressing *distributions* button, you can

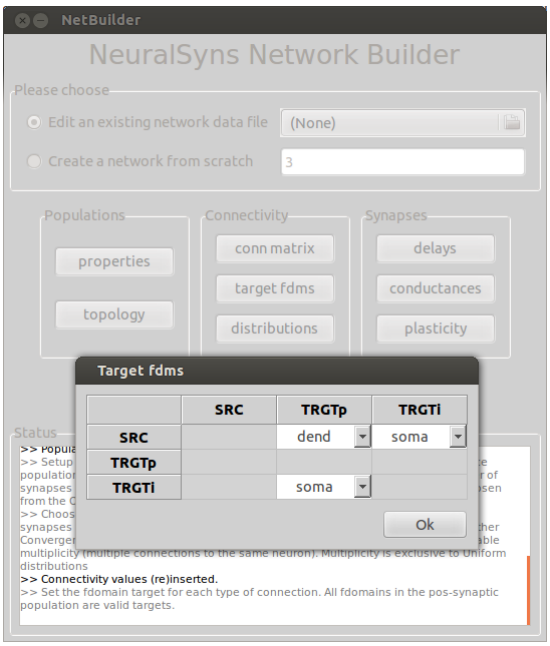

Figure 9: NetBuilder target functional domains.

define the connectivity volumes for each type of connection using a 3D Normal distribution. Note that normal distributions are only considered if $\sigma \leq size * d\_mean$. Independent Uniform distributions are used instead in the directions of space where the standard deviation values are large. For our model change the standard deviation values in the connection pair "SRC-TRGTi" for the values $\sigma_x = 10$, $\sigma_y = 10$ and $\sigma_z = 10$. Leave the other distribution values as their default values (see Fig 10).

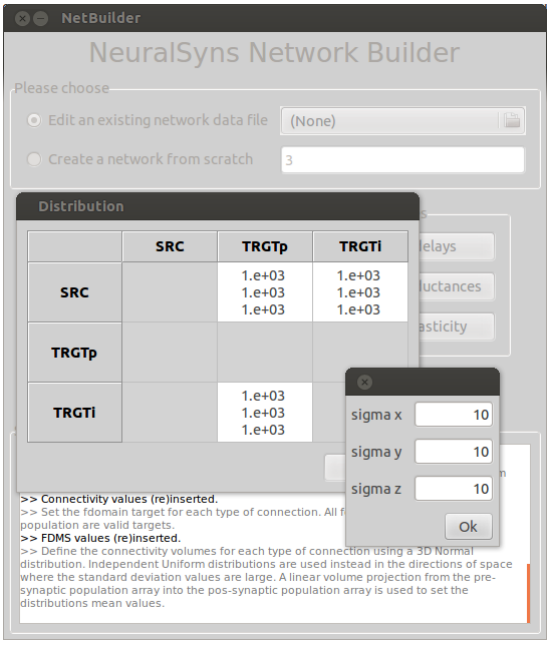

Figure 10: NetBuilder receptive fields distribution.

Press the *Ok* button to return to the main frame. Note that in the *Status* textbox, a warning message appears indicating the distribution pairs that were considered Uniform.

1.2.3 Synapses

We now move to the following frame *Synapses*, which rely on the previous defined main connectivity matrix to identify and parametrize all different types of synaptic connections. Here we can start

by pressing *delays* button in order to set the synaptic delay between connections, that is the delay between the pre-synaptic neuron spike time and the time of initiation of the post-synaptic conductance response. This time delay is therefore the sum of axonic transmission delay and synaptic transmission delay. As with the connectivity table, every entry in this table represents a specific fiber pathway. Assign $0.5ms$ for TRGTi-TRGTp time delay and the other two connection pairs delays to $2.0ms$ (Fig. 11).

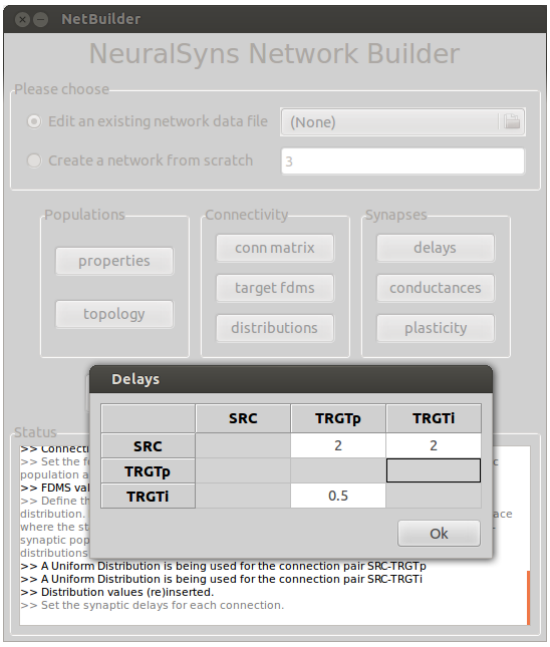

Figure 11: NetBuilder synaptic delays.

NeuralSyns segregates plasticity dynamics from the post-synatic response dynamics. For each class of dynamics, NeuralSyns includes the following conductance models:

- ISyn - Synapse with constant current injection, during a predefined time window.
- IExpSyn - Synapse with exponential current injection.
- ExpSyn - Synapse with conductance step increase and exponential conductance decay.
- Exp2Syn - Synapse with dual exponential conductance profile.
- AlphaSyn - Synapse with alpha conductance profile.
- NMDA - NMDA receptor with dual exponential conductance profile and magnesium block dynamics.
- AMPA+NMDA - AMPA and NMDA receptors combined.

Now, press *conductances* button and assign a model to the connection pairs already defined. When selected, the GUI gives access to new windows (Fig. 12) where the required parameters can be set. In our model set ExpSyn to the connection pair SRC-TRGTp, changing the conductance *g_step* to $0.03\mu S$; set ExpSyn to the connection pair SRC-TRGTi, changing the conductance *g_step* to $0.01\mu S$; and set Exp2Syn to the pair TRGTi-TRGTp and changing the conductance *g_max* to $0.01\mu S$, the reversal potential E to $80mV$, the *time_rise* to 1 ms and the *time_decay* to 10 ms.

The most interesting synapses in the nervous system are not static; instead, their efficacy - measured as the synapse's isolated contribution to induce post-synaptic firing - can change through time. These changes can last for either a short amount time or persist for long period of time. NeuralSyns include the following plasticity models:

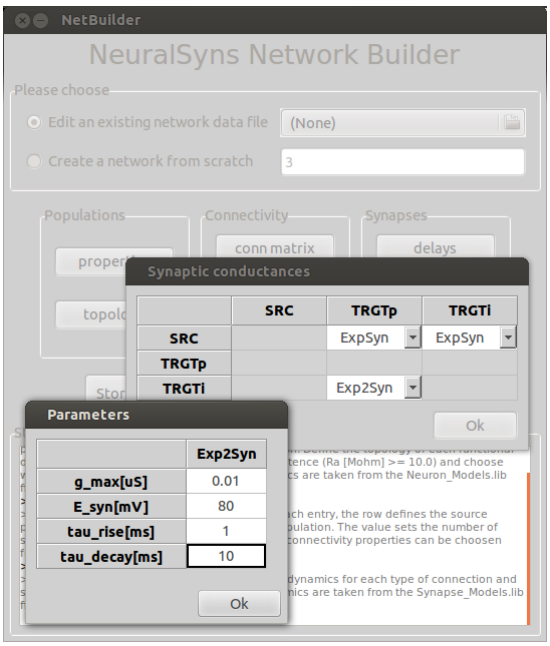

Figure 12: NetBuilder synaptic conductances models.

- Static - This is a dummy model to implement static synapses.
- STP - A short-term synaptic plasticity model - modified from Fuhrmann et al (2001)[1].
- SDP - A nearest neighbor spike-timing dependent plasticity model.
- Heeb - A standard Hebbian plasticity model.

Here we will explore NeuralSyns's representation of a class of synapses with short-term plasticity. In these synapses, the efficacy changes last for small periods of time, in the order of seconds or minutes. NeuralSyns represents dynamical synapses using the probability model of Fuhrmann. Briefly, the conductance of a dynamical synapse is modulated by a multiplication factor, $\in [0, 1]$. This factor results from the interaction of two competing mechanisms: facilitation and depression. Three parameters define a dynamical synapse: recovery time constant τ_{rec} [ms], facilitation time constant τ_{fac} [ms] and step increase U_1 [1]. NeuralSyns lets you define all of these parameters.

Click on the *plasticity* button and select the STP model for the connection pair SRC-TRGTi. In the parameters frame type "1000.0" in the "time_fac" entry, "0.1" in the "tau_rec" entry and "0.3" in the "U_1" entry (Fig. 13). Leave the other two connection pairs as *Static*. After confirming in the *Ok* button and returning to the main frame our model is ready to be created.

At this stage all the parameters required to define our large network model have been set. It is worthwhile to store all our model parameters in a data file. These .dat files can be loaded by NetBuilder and used to edit or change models with many parameters specified, saving the trouble of having to enter all data again. Save your model by pressing the *Store data file* button in the lower left of the NetBuilder main frame. Save with the name "3pop.dat" inside your working directory. This file, however, is not the file that we can use to simulate our model. To produce such file press the *Create .net file* button, also in the lower center of the NetBuilder main frame. Save your network model file with the name "3pop.net". Large models can take a considerable amount of time to be converted to a .net format. You can check the progress in a progress bar.

Finally, by pressing *Summary* button, you can have access to all the network information summarized in a new frame like fig. 14. This information can be accessed any time during the network creation.

[1]Fuhrmann G, Segev I, Markram H and Tsodyks M. *Coding of temporal information by activity-dependent synapses. J Neurophysiol*, **87**(1):140-148. (2002)

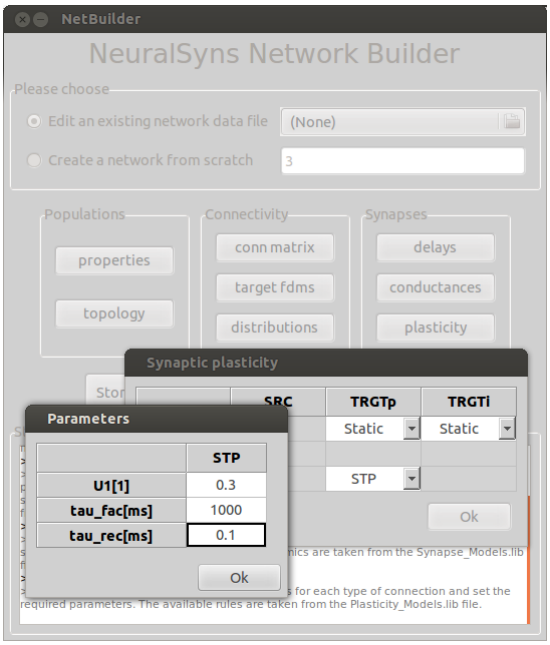

Figure 13: NetBuilder synaptic plasticity models.

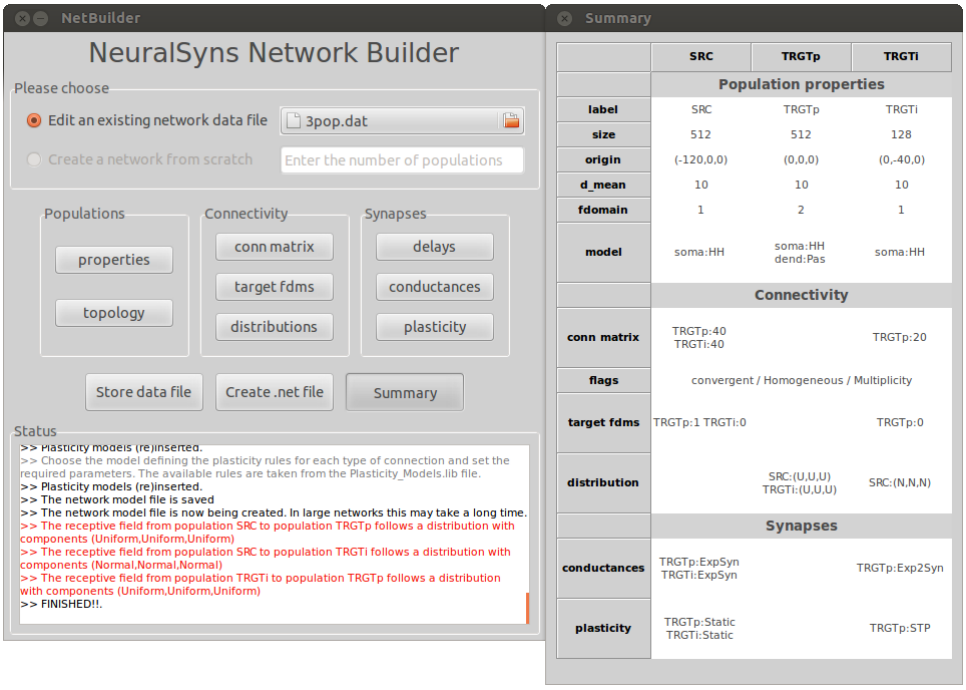

Figure 14: NetBuilder summary frame.

2 User defined stimulation protocols

The network model files contain all the information specifying a model but do not include entry points to specify stimulations and recordings of the network activity. The same network model can be simulated with different protocols. The NeuralSyns structure requires an electrode file (.elec) with the stimulation and recording protocols. The electrode files can be generated manually from a template or from another graphical tool provided with NeuralSyns, the *ElectrodeBuilder*. An .elec file consists of a list of time intervals, target neurons and stimuli/recording type and parameters. The file is called in NeuralSyns using the -e option followed by the name of the stimuli file. A self-explanatory .elec template file ("template.elec") is produced when NeuralSyns is called without any arguments. Check this file to see the stimuli options available by default in NeuralSyns.

The .elec file is a structured ASCII file where each line sets an new electrode with the following structure:

```
<elect0 start time [ms]> [<first neuron id>, <last neuron id>] <electrode id> <paramenters par0 par1 par2 ...>
<elect1 start time [ms]> [<first neuron id>, <last neuron id>] <electrode id> <paramenters par0 par1 par2 ...>
...
```

To access ElectrodeBuilder, open a shell, change to your working directory and run the binary ElectrodeBuilder:

```
>> ElectrodeBuilder
```

In the *Electrode* entry of the ElectrodeBuilder main frame you will find several options of electrodes that allow neuron stimulation and neuron recordings:

- injFire - fire neurons in range
- injFirePeriodically - fire neurons periodically
- injFireRandom - fires a selected number of random neurons
- injFirePoissonAsync - fire neurons independently, Poisson process
- injFirePoissonSync - fire neurons synchronously, Poisson process
- injCurrentConstant - inject constant current
- injCurrentGaussianNoise - inject random current (Gaussian)
- injCurrentSine - inject sinusoidal current
- recV0 - record membrane potential from fdm00
- recVAll - record membrane potential from all fdomais
- recStates0 - record state variables from fdm00
- recStatesAll - record state variables from all fdomains
- recSynStates - record state variables from all synapses

To define a stimulus protocol for the created network, select the electrode *inject random current (Gaussian)* which allows you to inject current generated by a Gaussian noise. A parameter frame appears where you need to define mean and standard deviation of the Gaussian distribution and the duration of the stimuli. Set "mean" entry to 2 nA, "stdev" to 0.2 nA and "duration" to 500.0 ms (Fig. 15).

In the *Time* entry of the ElectrodeBuilder frame you must insert the time relative to the electrode activation. Set time to 10 ms. In the *Range* entries you must define the range of neurons subject

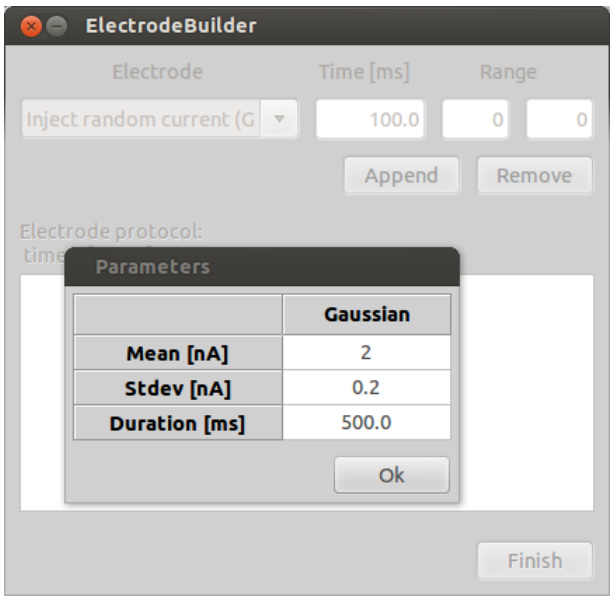

Figure 15: ElectrodeBuilder frame: define stimulation protocols parameters.

to the electrode. Since we have 512 neurons in SRC population we can set range with 0 and 511. Remember that the neurons are counted starting in 0. By pressing *Append* button you are including this protocol into the electrodes list.
Follow the same procedure to append a recording protocol. Select in the *Electrode* entry the option *record membrane potential from all fdomains*, which means we want to record the membrane potential for all fdomains of the three cells of the TRGTp population neurons. In the parameters frame you can define the periodicity and the duration of recording. Leave the "periodicity" entry with 1 and set "time" entry to 1000.0 ms. (Fig. 16)

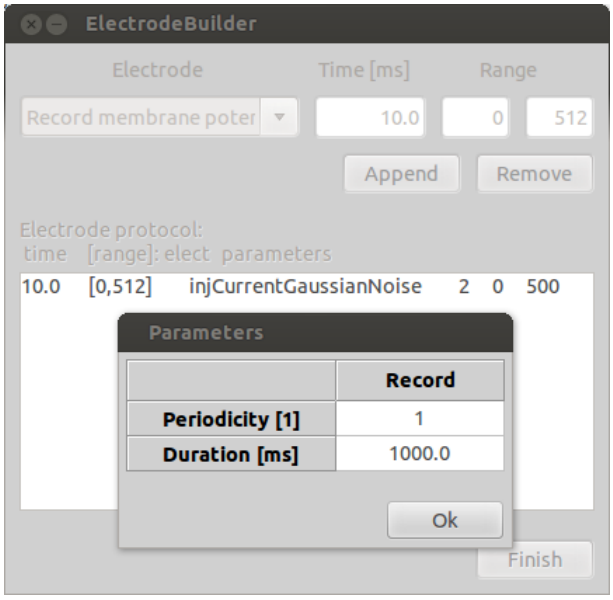

Figure 16: ElectrodeBuilder frame: define recording protocols parameters.

In Time entry set 0.0ms to start recording in the beginning of the simulation, and in the Range entries set 0 and 1151 (is the total on neurons in the three populations). Press *Append* to add also this protocol to the electrodes list. Note that in the lower part of the EletrodeBuilder frame the introduced electrodes appears in a textbox status, in the same structure as will appear in the electrode file. To help, stimulation electrodes have the 'inj' prefix while recording electrodes have the 'rec' prefix. (Fig. 17) The *Remove* button allows you to remove the last appended electrode. Finally,

press *Finish* button to create the electrode (.elec) file. Save as *3pop.elec* in your working directory.

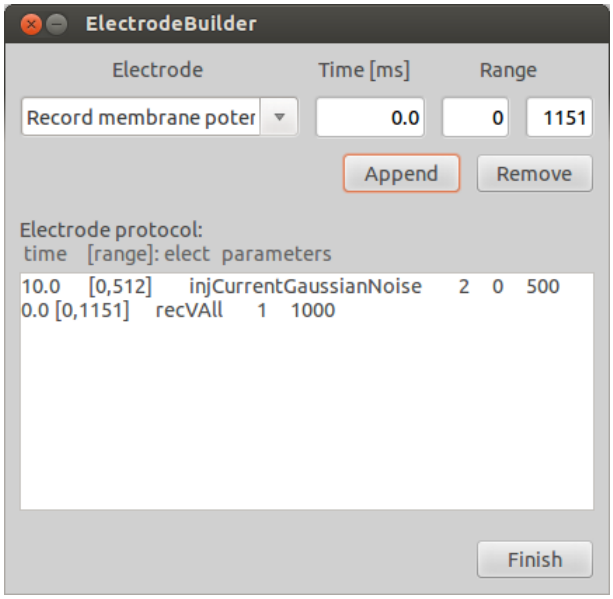

Figure 17: ElectrodeBuilder main frame: list of inserted electrodes.

We are now ready to use NeuralSyns, to run the simulation of this network model and assess the behavior of each neuronal population.

3 Simulation

A typical simulation with NeuralSyns consists in calling the simulation engine specifying the .net network model, the .elec electrodes file and the simulation time, as described below:

```
>> NeuralSyns 3pop.net -e 3pop.elec -t 1000.0
```

where the .elec file should be proceeded by "-e" flag. Simulation time is defined with the flag "-t" followed by time in milliseconds.
Optionally, since stochastic activation stimuli is frequently used in simulations, there is an implemented option in Neuralsyns command line: the option -S forces all the neurons in the first population to fire spontaneously and stochastically; the option -S must be followed by a positive number defining the average firing rate, in Hz. The interspike times follow a Poisson distribution. Without using the .elec file you can run the simulation with the following command line:

```
>> NeuralSyns 3pop.net -S 10.0 -t 1000.0
```

Before proceeding, take some time exploring the other available simulation options. Type NeuralSyns without any arguments, for listing all the options.

3.1 OpenGL graphical window

In many situations, it may be quite helpful to have a graphical visualization of the architecture and dynamics of our network model. NeuralSyns provides such an option, which is activated with the "-g" flag in the command line. Let us explore a bit this option and use it to check if the architecture of our 3pop.net model is as intended. Run the model with NeuralSyns using only the flag "-g". This will only display the network model.

```
paulo@brain$ NeuralSyns 3pop.net -g
```

NeuralSyns OpenGL visualization window has 3 modes: schematic, sphere and dot. By default you start with the "schematic" mode (18 left) which discards all 3D information and displays all neurons in a computationally efficient manner. All neurons are grouped by populations and their color is color coded for membrane potential: gradient of blue for hyperpolarized, gradient of red for depolarized, white for resting potential and yellow for firing neurons. In this model, all our neurons start at resting potential so all are white colored. The color code is the same for all visualization modes.

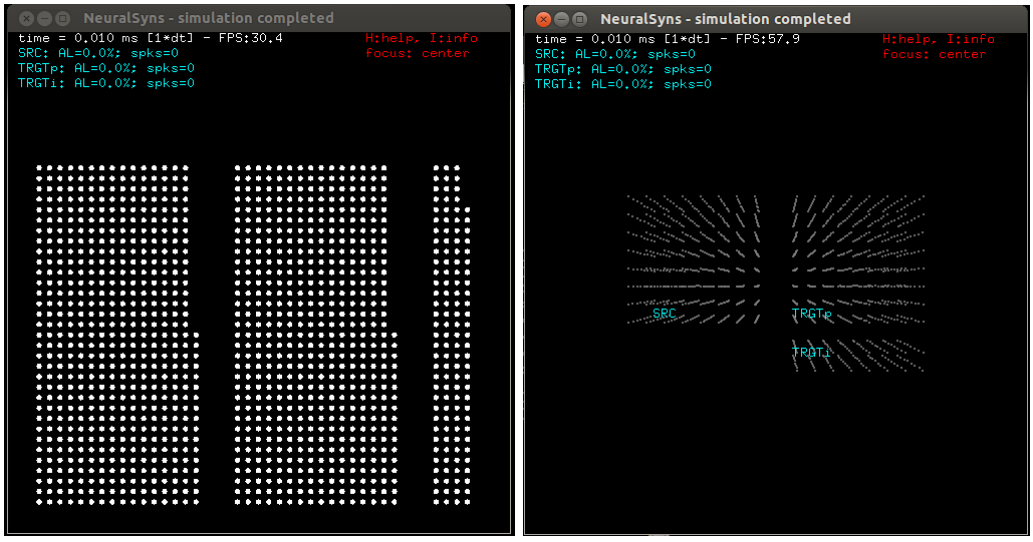

Figure 18: NeuralSyns - OpenGL graphical window: schematic mode at right and dot mode at left

To infer if the model architecture is as intended, change to the "sphere" visualization by pressing the mouse middle button. Note: we made some topology modifications (soma size and fdomain position)

in population TRTGp in order to visualize better the "dend" fdomains. You should get something similar to Fig. 19. Maximize the graphical window to have a bigger image. A list of options is presented to you if you press "h". Press "C" to toggle all connections. Use the navigation keys listed when you press "h" to explore the network model architecture. Navigate with the connections turn off and, if required, switch to the "dot" visualization mode (18 right) which improves performance by reducing the detail of the neurons representation.

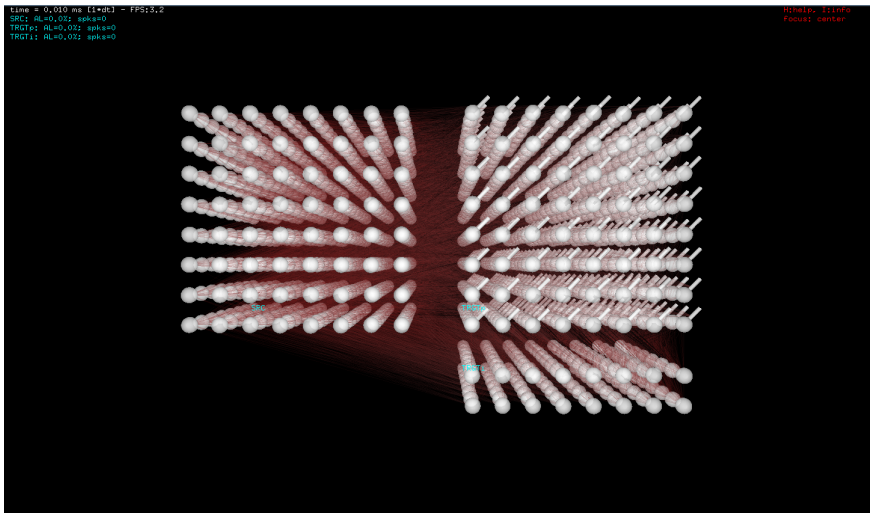

Figure 19: NeuralSyns: OpenGL graphical window

Now let us simulate the model using the electrode file (3pop.elec). Run NeuralSyns with the following commands:

```
paulo@brain$ NeuralSyns 3pop.net -e 3pop.elec -t 2000 -g -K -A
```

Option "-K" stores neuron's spike times in 'out_spike_times.dat', at each time step, and option "-A" stores activity levels in "out_activity_levels.dat" at each time step. During the simulation, use "p" to pause, and any other option you may want to try from the list - type "h" (see Fig. 20). Check how the schematic color code is so helpful during the simulation.

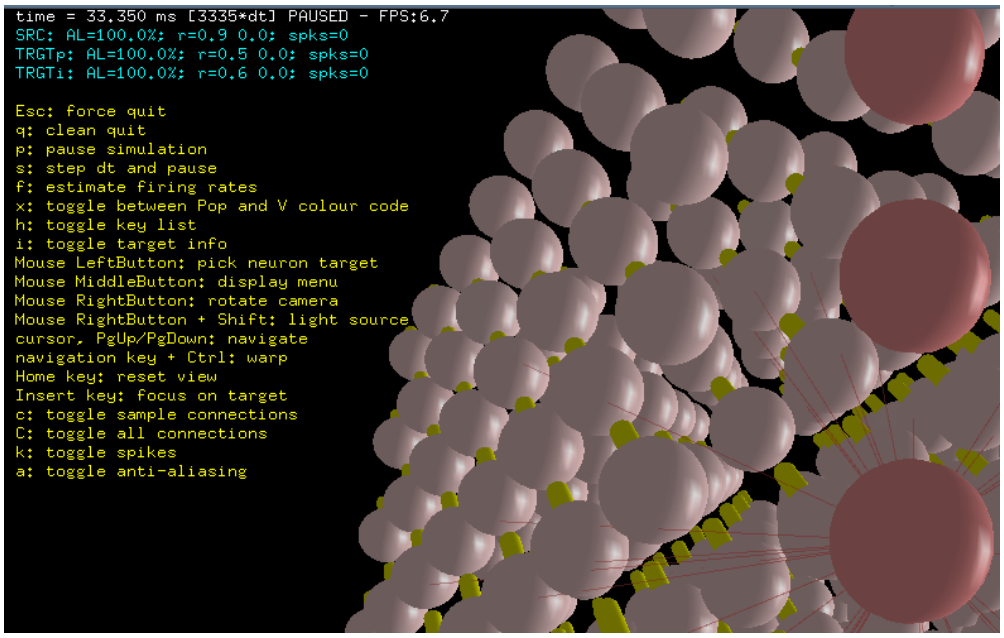

Figure 20: NeuralSyns OpenGL graphical window

Important Note
The peak conductance values defined in this model were chosen so that, given a pre-synaptic activity level, specific activity levels would be produced at the post-synaptic population. Several parameters have to be taken into account to calculate the peak conductances: 1) integrate-and-fire model properties V_{rest}, V_{thresh}, τ_m and R_m, 2) post-synaptic conductance profiles and 3) source activity and target activity represented in terms of the number of activated neurons in the population.

3.2 Data analysis

NeuralSyns does not incorporate data analysis tools but the output files use standard formatting and can be straightforwardly imported to specialized software, such as Scilab, Octave or Matlab, for later analysis and plotting. In the NeuralSyns directory you can find a *Tools-DataAnalysis* folder with several Matlab scripts that allows you to directly open the recorded data. When the simulation is finished, go to Matlab command line (to DataAnalysis folder) and type one of the following commands:

`>> NS_PlotV`

Plot the membrane potentials across time. Loads the "out_elec_2_recStatesAll.dat", that was produced by the recording electrode (defined in section 2).

`>> NS_PlotActivity`

Plot the activity level of all populations across time. The activity level is defined in NeuralSyns has the fraction of neurons in the population that fired in last 10.0 ms (which is in the same order of magnitude as the passive membrane time constant). These values are stored in "out_activity_levels.dat" file when calling "-A" flag.

`>> NS_RasterPlot`

Plot a raster plot with all firing times. Loads the "out_spike_times.dat" file created with the flag "-K".

`>> NS_ISI_Stats`

Calculate inter-spike interval means and standard deviations of firing times. Also, loads the "out_spike_times.dat" file created with the flag "-K".

`>> NS_PlotStates`

Plot state variables recorded in a specific recording protocols (see the list of protocols in section 2 for more details).

All NeuralSyns data output files are text files, which mean that you can use your favorite data analysis software instead of Matlab.
From all the output graphs you can easily interpret and critically assess your results. There is lots of information in our network model. Also, explore by yourself other stimuli and try to make the model more robust to excessive excitation in population SRC. Try a bit of "neuronal engineering" by changing the model parameters in NetBuilder.

4 Python interface

NetBuilder is a generic builder, providing access to the most common network model properties and dynamics available in Neuralsyns. But NeuralSyns does not cover all possibilities. You may want to specify higher level properties, such as dependencies between the basic parameters, which are not available from NetBuilder's GUI. These situations are accounted for in NeuralSyns by a python language interface that allows the fine-tuning, or more substantial modification, of the network model. Inside your NeuralSyns tools folder you can find the python interface in the *PyEditNet* folder. You can add/create whatever functions you may feel necessary to edit your network. But there are at least to main functions that make the bridge between NetBuilder and Python:

1. **Read_Network**('network.net') - Reads all data from the network model data file and load the full network model representation into python classes.
2. **Write_Network**('network.net') - Writes a network model data from the python environment into a .net file.

The python structure of classes used to store the network model is described in the NeuralSyns documentation (see "structures_list_python.doc").

In addition to the core read and write functions, the python interface also includes some example functions for manipulating the network:

- **Edit_Axon_Delay**(pop_source, pop_target, velocity) - Calculate new synaptic axon delays. The velocity parameter is assumed to be in $\mu m/ms$ and is used to convert distances to time delays. The parameter *pop_source* is the source population ID, *pop_target* is the target population ID and *velocity* is the action potential propagation velocity in this connection. It calculates new synaptic delays for each synapse.
- **Produce_Synaptic_Gaussian_Profile**(pop_source, pop_target, wfactor, sigma) - Produce a synaptic gaussian profile, by changing the weights of the connection from each neuron of the source population *pop_source*, using a weight factor and a standard deviation for the gaussian profile. The parameter *pop_target* is the population target ID, *wfactor* is the weight factor and *sigma* is the standard deviation of Gaussian distribution.
- **Define_Interception_Volume_Topology**(nrn, pop_target, 'shape', $args[]$)- Defines an interception volume topology for a given neuron ID and a target population. The parameter *nrn* is the neuron source ID, *pop_target* is the population target ID and *'shape'* is the name of the volume used as topology ('cubic', 'ellipsoid', 'cone' or 'cylinder'). For each shape the argument $args[]$ represents:

  cubic $-> args = [a, b, c]$ [length, weight, high]

  ellipsoid $-> args = [a, b, c]$ [radius in axis x, radius in axis y, radius in axis z]

  cone $-> args = [r, h]$ [base radius, high]

  cylinder $-> args = [r, h]$ [base radius, high]

- **Plot_RF**(nrn, pop_target) - Plots the connection volume topology of one neuron to one target population. Parameter *nrn* is the source neuron ID and *pop_target* is the target population ID.

To show the use of the python interface consider the 3pop network already built. Open a shell and type

```
$ pyEditNet 3pop.net
```

to open the python interface and automatically load the network model into the appropriate data structures.

One of the modifications that you can produce is relative to the 3D placement of your population neurons. You can give some anatomical properties to your model, by changing the shape of your populations. Try to change the shape of your inhibitory population (TRGTi) by typing the following command lines:

```
>>> for nrn in range(1023:1151):
>>>  neuron[nrn].x = f(x,y)
>>>  neuron[nrn].y = g(x,y)
```

where $f(x, y)$ and $g(x, y)$ are your's shape manipulating functions. The for cycle goes through all neurons belonging to the TRGTi population and change their position in space. Finally, type:

```
>>> Write_Network('3pop_modified.net')
```

that writes the modified network to a new .net file that can be used by new simulations in NeuralSyns. Explore the other python functions to edit your network. You must also create your own python functions for specific network modifications.

5 Adding new neuron and synaptic models to Neuralsyns

NeuralSyns would be extremely limited if it would not allow users to expand its set of dynamics by adding new models. NeuralSyns separates model dynamics into three classes:

- neuron dynamics
- synaptic dynamics
- plasticity dynamics

All models are written in C programming language and each model has its own .c code file. Every model previously mentioned to be included in basic form of NeuralSyns can be found inside *dynamics* folder, that is respectively divided into the three dynamic classes folders. For each model class, NeuralSyns has a template file which can be used to add new models. Two main constrains are imposed in these templates:

1. a formatted preamble comment with name of the model, parameters and state variables. For example, the Integrate-fire neuron model starts as:

```
 /*
MODEL DESCRIPTION:
LIF
SPIKE = yes
PARAM = 6:
Cm[nF] = 1.0
Rm[Mohm] = 20.0
Vrest[mV] = -70.0
Vthresh[mV] = -50.0
Vreset[mV] = -70.0
RefracP[ms] = 5.0
STATE = 0:
END
*/
```

2. the presence of three specific callback functions:
   - ModelInitialize_
   - ModelActivate_
   - ModelAdvance_

The formatted preamble is used to automatically generate model libraries which make the models visible to NetBuilder and the callback functions are used by NeuralSyns to incorporate the models into the network dynamics. NeuralSyns installation process is already prepared to automatically produce the model libraries for NetBuilder, compile and link the C code files.

Important Remarks

Keep all your model dynamics changes in the respective folders. This means they will be available in future session of NetBuilder. All NeuralSyns model data structures are available to "models.c". Populations data structures are defined in "populations.h", neurons data structures are defined in "neurons.h" and synapses data structures are defined in "synapses.h". All code is commented and *Doxygen* documentation can be produced by calling "doxygen" in ∼/NeuralSyns/doc:

```
paulo@brain$ cd ~/NeuralSyns/doc
paulo@brain$ doxygen
```