

iqr manual

for version 2.x

iqr.sourceforge.net

Ulysses Bernardet
bernuly@gmail.com

November 8, 2009

Contents

1	Introduction	5
1.1	Models in iqr	5
2	Working with iqr	8
2.1	Starting iqr	8
2.2	The User-interface	8
2.2.1	iqr Settings	10
2.3	Creating a new system	11
2.4	System properties	11
2.5	Opening an existing system	12
2.6	Saving the system	12
2.7	Creating processes	13
2.8	Process properties	13
2.9	External processes	13
2.10	Creating groups	15
2.11	Group properties	15
	Group neuron type	15
	Group topology	15
2.12	Connections	17
2.13	Creating connections	20
	Connection across processes	21
2.14	Specifying connections	21
2.14.1	Pattern	21
	PatternForeach	21
	PatternMapped	22
	PatternTuples	23
2.14.2	Arborization	24
2.14.3	DelayFunction	26

2.14.4 AttenuationFunction	27
2.15 Running the simulation	27
2.16 Visualizing states and collecting data	28
2.16.1 State Panel	28
2.16.2 Drag & drop	28
2.16.3 Space plots	29
2.16.4 Time plots	30
2.16.5 Connection plots	31
2.16.6 Data Sampler	31
2.16.7 Saving and loading configurations	33
2.17 Manipulating states	33
2.18 Support for work-flow of simulation experiments	35
2.19 Modules	36
3 Appendix	38
3.1 Neuron types	38
3.1.1 Random spike	38
3.1.2 Linear threshold	38
3.1.3 Integrate & fire	40
3.1.4 Sigmoid	42
3.2 Synapse types	44
Apical shunt	44
Fixed weight	44
Uniform fixed weight	44
3.3 Modules	45
3.3.1 Threading	45
3.3.2 Robots	45
Khepera and e-puck	45
3.3.3 Video	49
3.3.4 Lego MindStorm	50
3.3.5 Serial VISCA Pan-Tilt	51

1 Introduction

`iqr` is a tool for creating and running simulations of large-scale neural networks. The key features are: • graphical interface for designing neuronal models • graphical on-line control of the simulation • change of model parameters at run-time • on-line visualization and analysis of data • the possibility to connect neural models to real world devices such as cameras, mobile robots, etc. • predefined interfaces to robots, cameras, and other hardware • open architecture for writing own neuron, synapse types, and interfaces to hardware.

Every simulation tool is implicitly or explicitly driven by assumptions about the best approach to understand the system in question.

The heuristic approach behind `iqr` is twofold: Firstly we have to look at large-scale neuronal systems, with the connectivity between neurons being more important than detailed models of neurons themselves. Secondly, simulated systems must be able to interact with the real-world.

`iqr` therefore provides sophisticated methods to • define connectivity, • to perform high speed simulations of large neuronal systems that allow the control real-world devices – robots in the broader sense – in real-time, and • to interface to a broad range of hardware devices.

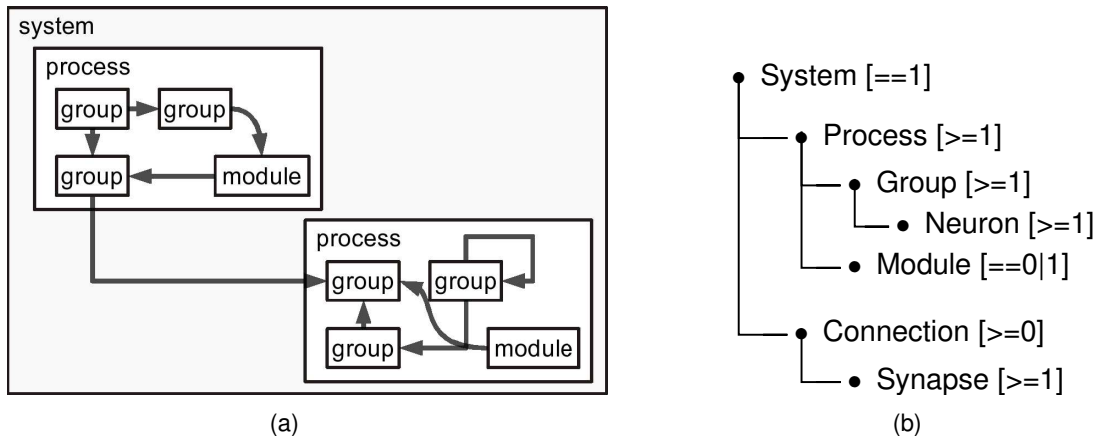
Simulations in `iqr` are cycle-based, i.e. all elements of the model are updated in a pseudo parallel way, independent of their activity. The computation is based on difference equation, as opposed to the approximation of differential equations. The neuron model used is a point neuron with no compartments.

As a simulation tool, `iqr` fits in between high-level, general purpose simulation tools such as Matlab® and highly specific, low-level, neuronal systems simulators such as NEURON. The advantages over a general purpose tool are on the one hand the predefined building blocks that allow easy and fast creation of complex systems, and on the other hand the constraints that guide the construction of biologically realistic models. The second point is especially important in education, where the introduction into a biological “language” is highly desirable.

Low-level simulators are not in the same domain as `iqr`, as in their heuristic approach, detailedness is favored over size of the model and speed of the simulation.

1.1 Models in `iqr`

Models in `iqr` are organized within different levels (fig. 1.1): the top level is the **system**, which contains an arbitrary number of **processes**, and **con-**

fig. 1.1: The structure of models in *iqr*

nections. Processes in turn consist of an arbitrary number of **groups**.

On the level of processes the model can be broken down into logical units. This is also the level where interfaces to external devices are specified.

A group is defined as a specific aggregation of neurons of identical type, specific in terms of the topology of the neurons in the group being a property of the group.

Connections are used to feed information from group to group. A connection is defined as an aggregation of synapses of identical type, plus the definition of the arrangement of the synapses.

framework:

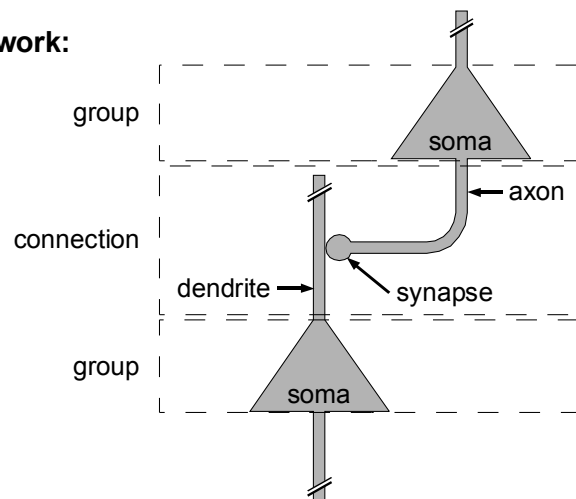


fig. 1.2: the group and connection framework

Figure 1.2 diagrams the simplest case of two connected neurons. It is important to notice which elements belong to which framework. Groups effectively only comprise of the neuron **soma**, where all inputs are summed

using the function of the specific neuron type (see Appendix Neuron types on page 38).



The connection framework deals with **axon**, **synapse**, and **dendrite** of the neurons. The computational element is the synapse which is calculating its own internal state, and the signal transmission according to the type (see Appendix Synapse types on page 44). More information about the connection concept can be found in section 2.12 on page 17.



The distinction between group and connection is not a biological fact per se, but an abstraction within the framework of `iqr` that allows an easier approach to modeling biological systems.


Conventions used in the manual

There are a few conventions that are used through out this manual: Text in a gray box refers to entries in `iqr`'s menu, and also to text on dialog boxes.

The symbol  marks passages in the manual that require special attention, hints and tips are marked with the symbol .

2 Working with iqr

2.1 Starting iqr

To start `iqr` enter '`iqr421`' at the prompt in a terminal window or double-click on the icon  on the desktop.

The table below lists the command-line options `iqr` supports:




<code>-f <filename></code> <code>--file <filename></code>	open system file <code><filename></code>
<code>-c <filename></code> <code>--config <filename></code>	open configuration file <code><filename></code> (see section 2.16.7 on page 33)
<code>-r</code> <code>--run</code>	automatically start simulation
<code>-v</code> <code>--version</code>	show version

The arguments `-f <filename>`, `-c <filename>`, and `-r` can be combined arbitrarily, but `-r` only works in combination with `-f <filename>`.

2.2 The User-interface

Diagram editing pane

The main **diagram editing pane** (fig. 2.1①) serves to add processes, groups and connections to the model. The definition of a new process automatically adds a new tab and therein a new diagram. To switch between diagram panes use the **tab-bar** (fig. 2.1②). The left-most tab always presents the system-level.

On the diagram editing pane, a gray square  represents a process, a white square  a group, and a line with an arrow head  a connection.

A single process or group is selected by clicking on its icon in the diagram editing pane. To select multiple processes or groups, hold down the control-key while clicking on the icon.

Diagram editing toolbar

The functionality of the diagram editing toolbar (fig. 2.1③) is as follows:

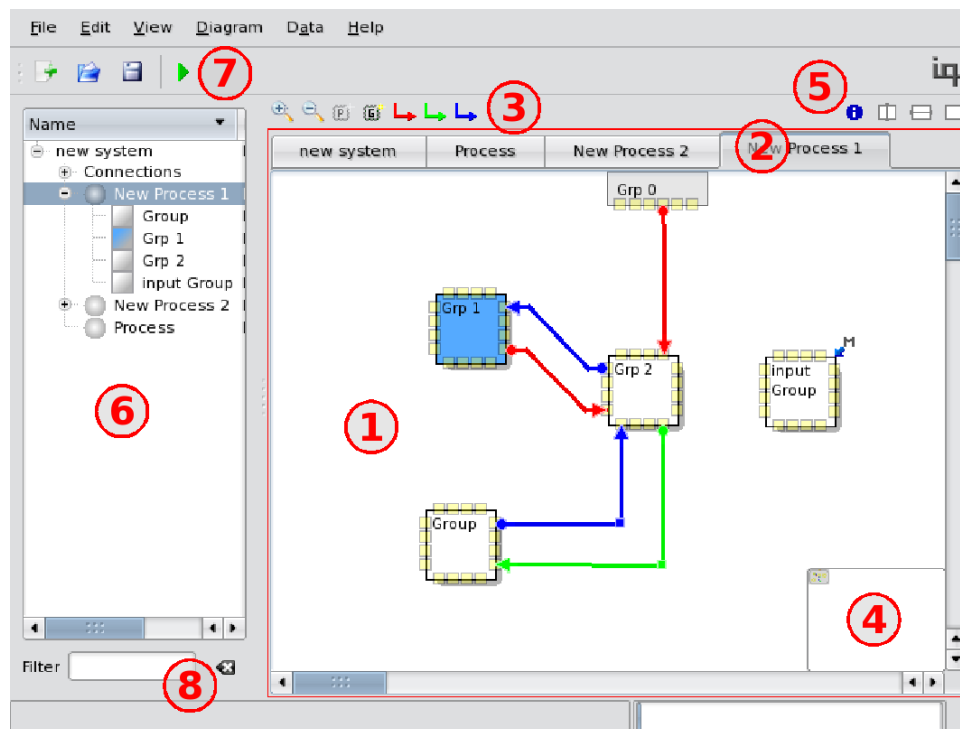


fig. 2.1: iqr graphical user-interface



Zoom in and out of the diagram



Add a new process to the system level



Add a new group to the current process



Add a new connection between groups: excitatory (red), modulatory (green), inhibitory (blue)

A more detailed description on how to use the diagram editing toolbar will be given when outlining the editing of the system.

Splitting the diagram pane



Split the diagram editing pane into two separate views by using the **splitter** (fig. 2.1 5). From left to right: split vertically, horizontally, revert to single window view.

Navigating the diagram

To navigate on a large diagram that does not fit within the diagram editing pane, the **panner** (fig. 2.1 4) can be used to change to the visible section of

the pane.

Browser

On the left side of the screen (fig. 2.1⑥) a tree-view of the model, the **browser**, can be found. It provides direct access to the elements of the system. The top node of the tree represents the system level (see 1.1), the second level node reflects the processes and the third level node points to the groups. By double-clicking on the system or process node you can open the corresponding diagram in the diagram editing pane. Right-clicking on any node brings up the context-menu.

Copy and Paste elements

You can copy and paste a process, one or more groups, and a connection to the clipboard. To copy an object, right-click on it and select **Copy ...** from the context-menu.

To paste the object, select **Edit** → **Paste ...** from the main menu. Processes can only be pasted at the system level, groups and connections only at the process level.

Print and save the diagram



You can export the diagram as PNG or SVG image. To do so, use the menu **Diagram** → **Save**. The graphics format will be determined by the extension of the filename. If you choose SVG as the export format, the diagram will be split into several files (due to the SVG specification). It is therefore advisable to save the diagram to a separate folder.

To print the diagram use the menu **Diagram** → **Print**.

2.2.1 iqr Settings

Via the menu **Edit** → **Settings** you can change the settings for iqr.

General

- **Auto Save** Change the interval at which iqr writes backup files. The original files are not overwritten, but auto save will create a new backup file, the name of which is based on the name of the currently open system file, with the extension `,autosave` appended.



The range is from `-1` to 60 minutes, where a value of `-1` disables auto save.


- **Font Name** Set the font name for the diagrams. The changes will only be effective at the next start of iqr.
- **Font Size** Set the font size for the diagrams. The changes will only be effective at the next start of iqr.

SynapsePath**ModulePath**

The options for NeuronPath, SynapsePath, ModulePath refer to the location where the specific files can be found. As they follow the same logic the description below applies to all three.

- **Use local ...** Specifies whether iqr loads types from the folder specified by **Path to local** below.
- **Use user defined ...** Specifies whether iqr loads types from the folder specified by **Path to user defined** below.
- **Path to standard ...** The folder into which the types were stored at installation time.
- **Path to local ...** The folder where system-wide non-standard types are stored.
- **Path to user defined ...** The folder into which the user stores her/his own types.

2.3 Creating a new system

A new system is created by selecting  from the main toolbar (fig. 2.1⑦) or via the menu **File** → **New System**. Creating a new system will close any open systems.

2.4 System properties

To change the properties of the system right-click on the system name in the browser (top-most node), and select **Properties** from the context-menu, or via the menu **File** → **System Properties**.

Using the system properties dialog (see fig. 2.2), you can change the name of the system, the author, add a date, and notes (**Hostname** has no meaning in the present release).

The most important property is **Cycles Per Second**, with which you can define the speed at which the simulation is updated.

A value of 0 means, the simulation is running as fast as possible, values larger than zero, denotes how many updates per cycle are executed. The value entered reflects “best effort” in two ways. On the one hand, the simulation cannot run

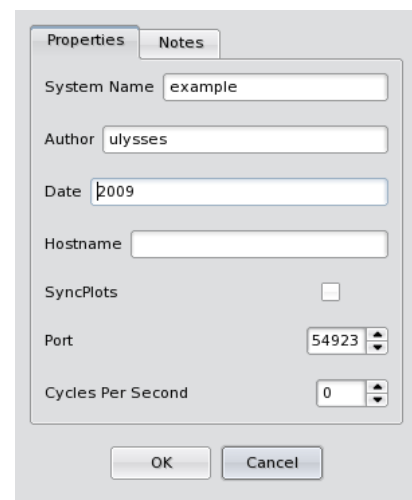




fig. 2.2: System properties dialog

faster than a certain maximum speed, as given by the complexity of the system, and the speed of the computer. On the other hand, slight variations in the length of the individual update cycles can occur.

2.5 Opening an existing system

An existing system is opened by selecting the button  from the main toolbar (fig. 2.1⑦) or via the menu **File** → **Open**. If you open an existing system, the current system will be closed.

2.6 Saving the system

To save the system, press the  button in the main toolbar (fig. 2.1⑦) or via menu select **File** → **Save System**. To save a system under a new name, select menu **File** → **Save System As**.

If your system contains inconsistencies, e.g. connections that have no source and/or target defined, a dialog as shown in fig. 2.3 will show up.



Please take this warning seriously. The system will be saved to disk, but you should fix the problems mentioned in the warning dialog or you might not be able to re-open the system.

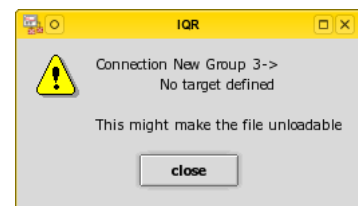




fig. 2.3: Warning dialog for saving invalid systems

2.7 Creating processes

To add a new process to the system, activate the system-level in the diagram editing pane by clicking on the left-most tab in the tab-bar (fig. 2.1②) or by double-clicking on the system node in the browser (fig. 2.1⑥). Thereafter, click the “Add Process” button  in the diagram editing pane toolbar (fig. 2.1③). The cursor will change to  and you can put down the new process by left-clicking in the diagram editing pane. To abort the action, right-click into any free space in the diagram editing pane.

2.8 Process properties

To change the properties of a process, either • double-click on the process icon in the diagram editing pane, or • right-click on the process icon or the process name in the browser, and select **Properties** from the context-menu.

Using the process properties dialog (see fig. 2.4), you can change the name of the process as it appears on the diagram, or you can add notes. The remaining properties in this dialog refer to the usage of modules and will be explained in section 2.19 on page 36.



After changes in the property dialog, you must click on **Apply** before closing the dialog, otherwise the changes will be lost.

fig. 2.4: Process properties dialog

2.9 External processes

In large-scale models it is not uncommon that several subsystems are combined into a larger model, and that certain circuits are used in multiple models. *iqr* supports this via the “external processes” mechanism; processes can be exported to a separate file, and linked or imported into an existing system. If a process is linked-in, it remains a separate file which can be worked with independently of system it is integrated in.

Exporting a process A process can be exported from an existing *iqr* system by right-clicking on the process icon in the diagram editing pane, or the process name in the *browser*, and choosing the menu entry **Export Process**. By default processes exported by *iqr* have the extension “.iqrProcess”.

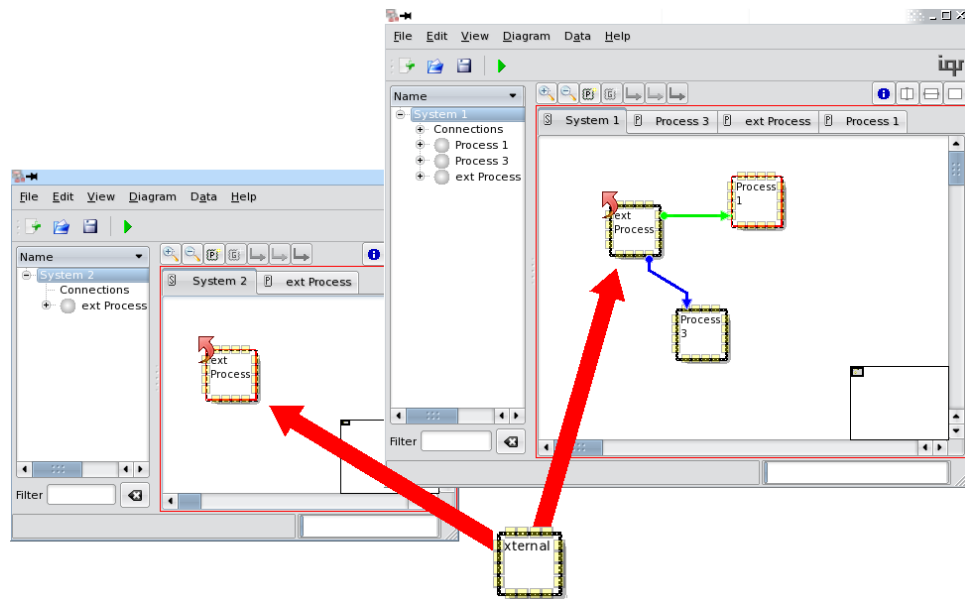




fig. 2.5: The concept of an “external process”: two iqr systems are including the same process, which is stored in a separate file.

Importing and linking-in a process A process stored in a separate file can be either *imported* or *linked into* an existing system. In the former case, the process is copied into the existing system, making the process stored in a separate file obsolete. In the case of linking-in a process, the definition of the process remains in a separate file, and is updated every time the system is saved. This also means that two or more iqr systems can include exactly the same process. Importing and linking are done via the menu **File** → **Import Process** and **File** → **Link-in Process**. In the case of a linked-in process, the path to the external process is shown in the properties dialog of the process.

2.10 Creating groups

To add a new group to a process, firstly activate the process diagram in the diagram editing pane by clicking on the corresponding tab in the tab-bar (fig. 2.1②) or by double-clicking on the process node in the browser (fig. 2.1⑥). Secondly, click on the button  in the diagram editing pane toolbar (fig. 2.1③). The cursor will change to  and you can put down the new group by left-clicking in the diagram editing pane. To abort the action, right-click in any free space in the diagram editing pane.

2.11 Group properties

To change the properties of a group, either • double-click on the group icon in the diagram editing pane, or • right-click on the group icon or on the group name in the browser, and select **Properties** from the context-menu.

You can change the name of the group as it appears on the diagram, or you can add notes by activating the group properties dialog (see fig. 2.6) .



After changes in the property dialog, you must click on **Apply** before closing the dialog, otherwise the changes will be lost.

Two additional group properties, the neuron type and the group topology, will subsequently be explained in more detail.

Group neuron type

By default, a newly created group has no neuron type associated to it. To select the neuron type for the group, take the following steps: • select the neuron type from the pull-down list (fig. 2.6①) • press the **Apply** button (fig. 2.6②). Now change the parameters for the neuron type by clicking on **Edit** (fig. 2.6③). An extended explanation of the meaning of these parameters for each type of neuron is given in the Appendix on page 38.

Group topology

The term *topology* as used in this manual refers to the packing of the cells within the group. The basic concept behind it refers to cells in a group being arranged in a regular lattice. To define the group's topology, proceed as follows: • select the topology type from the pull-down list • press **Apply** • press **Edit** .

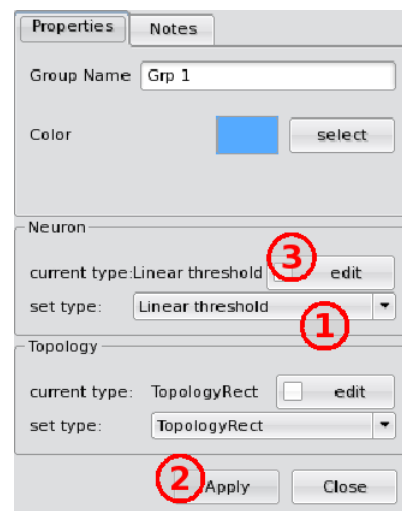


fig. 2.6: Group properties dialog

Topology types If we refer to a *TopologyRect*, every field in the lattice is occupied by one neuron. *TopologyRect* is therefore defined by the **Width** and the **Height** parameters. In case of *TopologySparse* the user can define which fields in the lattice are occupied by neurons. In fig. 2.7 the dialog to define the *TopologySparse* (a), and the graphical representation of the defined topology (b) are shown. To enter a new location for a neuron • select **add row** and • enter the *x* and *y* coordinates in the appropriate field of the table. To delete neurons • select the row you want to delete by clicking on it and • press **delete row**.



The coordinates of the lattice start at (1, 1) in the top-left corner.

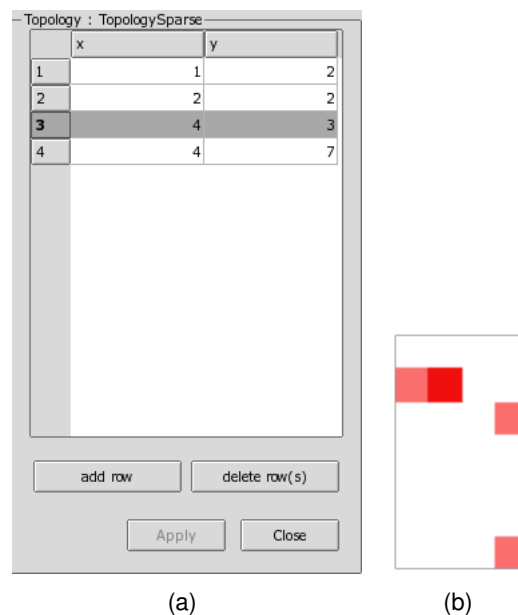


fig. 2.7: (a) Dialog to define *TopologySparse* (b) example of the topology

2.12 Connections

Groups and connections are described on two different levels. At the process level, groups and connections are created, and are symbolized as rectangles and lines with arrow heads respectively (see fig. 2.8(a)).

From figure 1.2 on page 6 we know, that groups are an abstraction of an assembly of neurons, and connections are abstractions of an assembly of axon-synapse-dendrite nexuses.

In terms of the structure the description at the process level is therefore underspecified. Regarding the group we neither know how many neurons it comprises of, nor what the topology – the arrangement of the neurons – is. Regarding the connection we don't know from which source neuron information is transmitted to which target neuron. The definition is only complete after specifying these parameters. Figure fig. 2.8(b) shows one possible complete definition of the group and the connection.

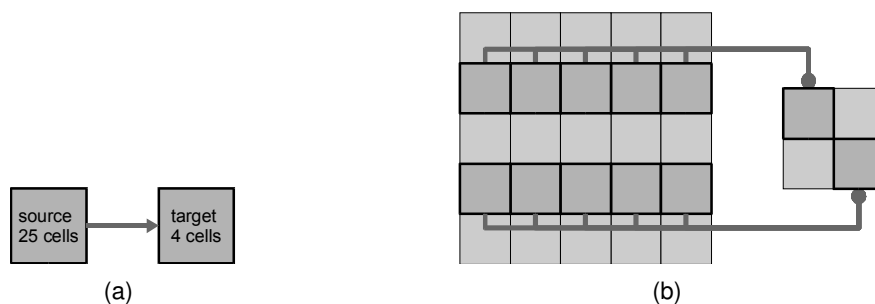


fig. 2.8: Levels of description for connections

In the framework of *iqr*, the following assumptions concerning connections are made:

- there is no delay in axons,
- the computation takes place in synapses,
- the transmission delay is dependent on the length of the dendrite,
- any back-propagating signals are limited to the dendrite.

In principle, the complete definition of a connection is twofold, in that it comprises of the definition of the update function of the synapse, and the definition of the connectivity. In this context, the term connectivity refers to the spacial layout of the axons, synapses and dendrites.

But update function and connectivity are not as easily separable, as the delays in the dendrites are derived from the connectivity.

Multiplicity As mentioned above, a single connection is an assembly of axon-synapse-dendrite nexuses. A single nexus in turn can connect several pre-synaptic neurons to **one** post-synaptic cell, or feed information from **one** pre-synaptic into multiple post-synaptic neurons.

A nexus can hence comprise of several axons, synapses, and dendrites.

In figure 2.9 two possible cases of many-to-one, and one-to-many are diagrammed. The first case (a) can be referred to as “receptive field” (RF), the second one as “projective field” (PF).

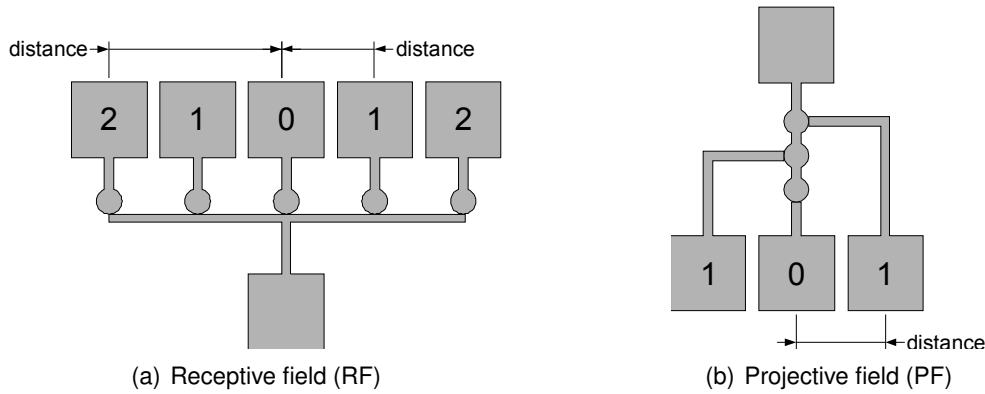


fig. 2.9: The axon-synapse-dendrite nexus and distance

For reasons of ease of understanding the connectivity depicted in fig. 2.9 is only one-dimensional, i.e. the pre-synaptic neurons are arranged in one row. This is not the standard case; most of the times a nexus will be defined in two dimensions as shown in fig. 2.10.

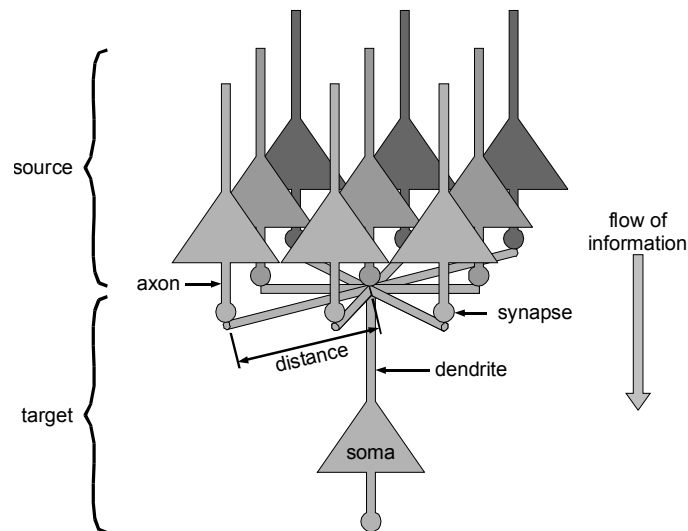


fig. 2.10: Two dimensional nexus

Delays The layout shown in fig. 2.9 is directly derived from the above listed assumptions that delays are properties of dendrites.

The basis of the computation of the delay is the distance, as given by the eccentricity of a neuron.

In the case of a receptive field, the eccentricity is defined by the position of

the sending cell relative to the position of the one receiving cell. In fig. 2.9a this definition of the distance, and the resulting values are depicted.

In a projective field, the eccentricity is defined with respect of the position of the multiple post-synaptic cells relative to the one pre-synaptic neuron (fig. 2.9(b)).

Patterns and arborizations

The next step is to understand how the connectivity can be specified in `iqr`.

Defining the connectivity comprises of two steps:

1. define the `pattern`
2. define the `arborization`

Pattern The *Pattern* defines **pairs of points** in the lattice of the pre- and post-synaptic group. These points are **not** neurons, but (x,y) coordinates. In fig. 2.11 the points in the pre-synaptic group are indicated by the arrow labeled “projected point”. For sake of ease of illustration we’ll use a one dimensional layout again. The groups are defined with size 10×1 for source and 2×1 for target.

The two pairs of points are:

pair 1: $(3, 1)_{pre}, (1, 1)_{post}$

pair 2: $(8.5, 1)_{pre}, (2, 1)_{post}$

Note that in the second pair the point in the pre-synaptic group is not the location of a neuron.

`iqr` provides several ways to specify the list of pairs. These different methods are referred to as pattern types. The meaning of the pattern types and how to define them is explained in detail below.

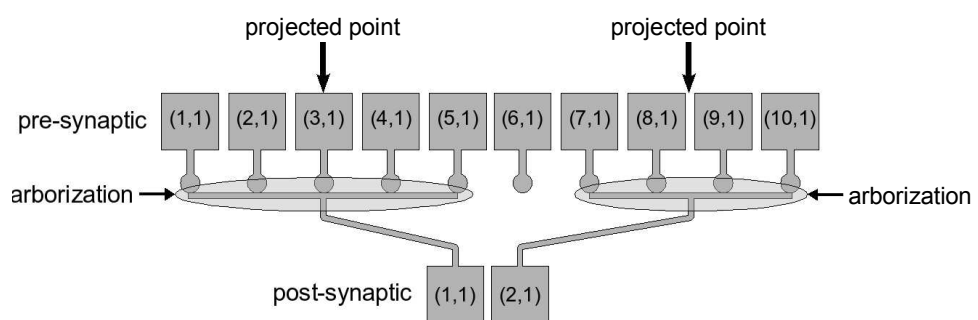


fig. 2.11: Nexuses, pattern, and arborization

Arborization As previously mentioned, the pattern does not relate to neurons directly, but to coordinates in the lattice of the groups. It is the arborization that defines the real neurons that send and receive input. This can be imagined as the arrangement of the dendrites of the post-synaptic neurons, as depicted in fig. 2.11. The arborization is applied to every pair of point

defined by the pattern. Whether the arborization is applied to the pre- or the post-synaptic group is defined by the direction parameter. If applied to the source group, the effect is a fan-in receptive field. A fan-out, projective field, is the result of applying the arborization to the target group (see also fig. 2.9).

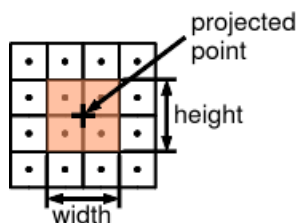


fig. 2.12: Rectangular arborization

iqr provides a set of shapes of arborizations. Figure 2.12 shows an example of a rectangular arborization of a width and height of 2. The various types of arborizations are described in detail below.

Combining pattern and arborization The combination of pattern and arborization is illustrated in fig. 2.13. The source group is on the left side, the target group on the right side. In the lower part you find the pattern denoted by a gray arrow. The pattern consists of four pairs of points. On the upper left tier, the arborization is applied to the points defined by the pattern. For each cell that lies within the arborization, one **synapse** is created. In the case presented in fig. 2.13 there are 16 synapses created.

For each synapse the **distance** with respect to the pattern point is calculated. This distance serves as basis to calculate the **delay**. Several functions are available, for details see the *DelayFunction* section further below.

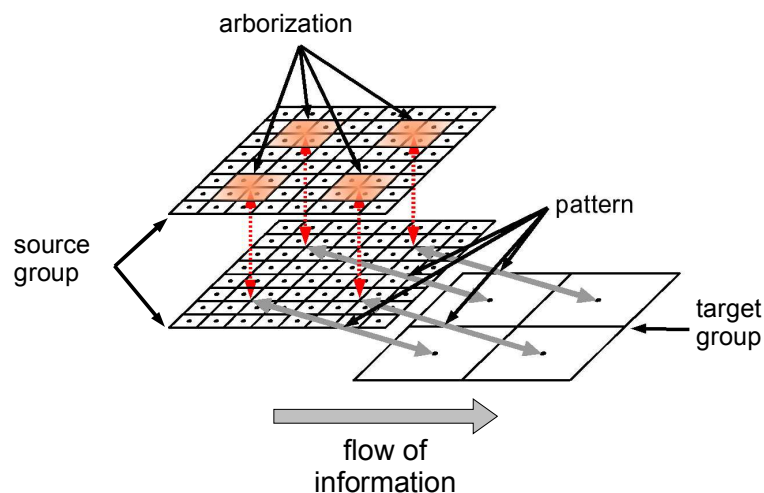





fig. 2.13: Combining pattern and arborization

2.13 Creating connections

To add a new connection, click on the corresponding button in the diagram editing pane toolbar (fig. 2.1③).  (red arrow) will create an *Excitatory*,  (green arrow) a *Modulatory* and  (blue arrow) an *Inhibitory* connection.

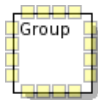





fig. 2.14:

After clicking on one of the buttons to create a connection, you will notice the cursor changing to . To position the connection, first click on one of the **yellow squares** (fig. 2.14) at the edge of the source group icon and then on one of the **yellow squares** at the edge of the icon for the target group.

To cancel the creation of a connection, right-click into any free space in the diagram editing pane.

Adding and removing vertexes To add a new vertex to the connection, hold down the ctrl-key and click on the connection. To remove a vertex from a connection, right-click on it and select **delete** from the context-menu.

Connection across processes

Connecting groups from different processes requires two preparatory steps. First you need to split the diagram editing pane into two views, by clicking on the split vertical  or split horizontal  button from diagram editing toolbar (fig. 2.1⑤). Secondly you will use the tab-bar (fig. 2.1②) to make each separate view of the diagram editing pane display one of the processes containing the groups you want to connect. Now you can connect the groups from the two processes just as described above.

Upon completion, you can return to the single window view by clicking on



After connecting groups from different processes, you will notice a new icon as shown in fig. 2.15 at the top edge of the diagram. This “phantom” group represents the group in the process currently not visible, which the connection targets to or originates from.

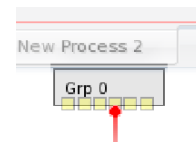


fig. 2.15:

2.14 Specifying connections

Via the context-menu or by double-clicking on the connection in the diagram, you can access the properties dialog for a connection.



Click on the connection line, not the arrowhead.

In the properties dialog, you can change the name and the notes for a connection as well as the **connection type**.

To select and edit the **synapse** type, the same procedure applies as when setting and editing the neuron type for the group (section 2.11 on page 15).

2.14.1 Pattern

This section explains the meaning of the different types of patterns and how to define them in iqr.

PatternForeach

This pattern type defines a full connectivity, where each cell from one group receives information from all the cells of the other group as shown in fig. 2.16. You can select all cells, limit the selection to a region, or define a list of cells.

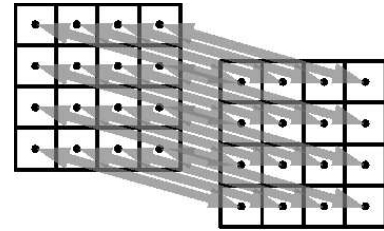


fig. 2.16:

Property dialog Figure 2.17 shows the dialog to set the properties for the Pattern-Foreach. To define which cells from the source and target group are included, use the pull-down menu ①, and select from **All**, **Region**, or **List**.

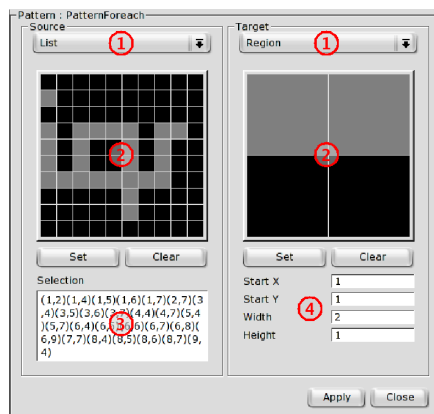


fig. 2.17:

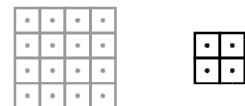
To define a **list** of cells, click on each cell you want to add to the list. Clicking **Set** will define the list, and the coordinates of the selected cells will be displayed in the **Selection** window ③. To clear the drawing area and to delete the list press **Clear**.

To define a **region**, click on the cell you want to be the first corner in the drawing area ②, move the mouse while keeping the mouse button pressed, and release the button when in the opposite corner. Once done, click in **Set** to define the region. The definition of the region will be shown in ④. To delete a defined region click on **Clear**.

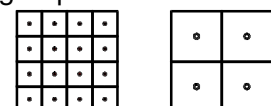
PatternMapped

In *PatternMapped* a mapping is used to determine the projection points. The procedure is as follows:

1. determine the small group (fewer neurons):



2. scale the smaller group to the size of the larger group:



3. determine the coordinates of the cells in the scaled group
4. apply these coordinates to the larger groups

Which group is mapped onto which depends solely on the size and not on which group is pre-, or post-synaptic. We refer to the surface covered by a

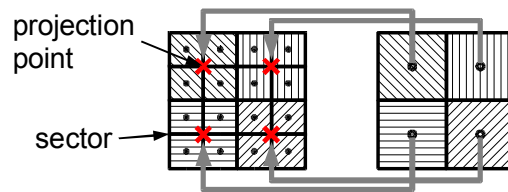


fig. 2.18:

neuron location in the scaled group, when overlaid over the larger group, as **sector**.



In fig. 2.18 the concept of the mapping is illustrated. As you can see from the depiction, projected points need not be at neuron positions.

Property dialog In the property dialog for *PatternMapped* you can select
 • All cells or a • Region of cells from the source and the target population to be used in the mapping. The procedure for setting the region is the same as explained for *PatternForeach*.

If the **mapping type** is set to **center**, only the central projection points are used, whereas if **all** is selected, all cells in the mapped sector are used.

PatternTuples

The *PatternTuples* serves to define individual cell to cell projections. You can associate an arbitrary number of pre-synaptic with an arbitrary number of post-synaptic cells.

A tuple t , as used in this definition, is the combination of n source cells with p target cells:

$$t_0 = \{(pre_0, \dots, pre_n), (post_0, \dots, post_p)\}$$

For the set of pre-synaptic and post-synaptic cells the Cartesian product will be calculated.

The pattern p itself is the list of tuples:

$$p = \{t_0, \dots, t_q\}$$

An example illustrates the concept. We have two tuples t_0 and t_1 :

$$\begin{aligned} t_0 &= \{(pre_0, pre_1, pre_2), (post_0)\} \\ t_1 &= \{(pre_2), (post_1, post_3, post_5)\} \end{aligned}$$

After the Cartesian product, the tuples are:

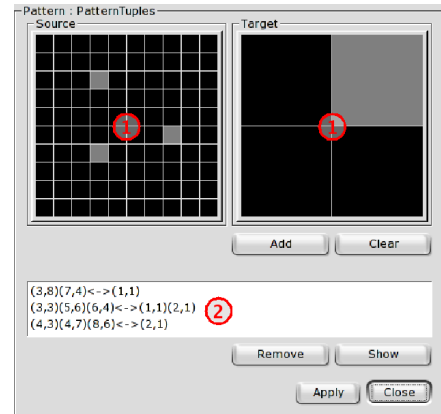
$$\begin{aligned} t_0 &= \{(pre_0, post_0), (pre_1, post_0), (pre_2, post_0)\} \\ t_1 &= \{(pre_2, post_1), (pre_2, post_3), (pre_2, post_5)\} \end{aligned}$$

The pattern p consists of the combination of the two tuples:

$$p = \{(pre_0, post_0), (pre_1, post_0), (pre_2, post_0), (pre_2, post_1), (pre_2, post_3), (pre_2, post_5)\}$$

Property dialog To define a tuple, select a number of pre-synaptic source cells, and a number of post-synaptic cells, by clicking on them in the respective drawing area ①. To add the tuple to the list click on **Add**. To clear the drawing area, click on **Clear**.

The list of tuples is shown in ②. You can remove a tuple by selecting it in ②, and pressing **Remove**. If you select a tuple, and click on **Show**, its corresponding cells will be shown in the drawing area ①. Now you can add new cells, and save the tuple as a new entry by clicking on **Add** again.



2.14.2 Arborization

In fig. 2.19 all available arborization types, bar *ArbAll*, are diagrammed. Where applicable the red cross indicates the projection point as defined by the pattern. The options available in the property dialogs for the different arborization types are derived from the lengths shown in the figure. In addition all types of arborizations have the parameters **Direction**, and **Initialization Probability**.

Property dialog The direction parameter defines which population the arborization is applied to: in the case of **RF** it is applied to the source group, in case of **PF** to the target group.

With the initialization probability you can set the probability with which a synapse defined by the arborization is actually created. E.g. a probability of .5 means, that the chance a synapses is created, is 50%.

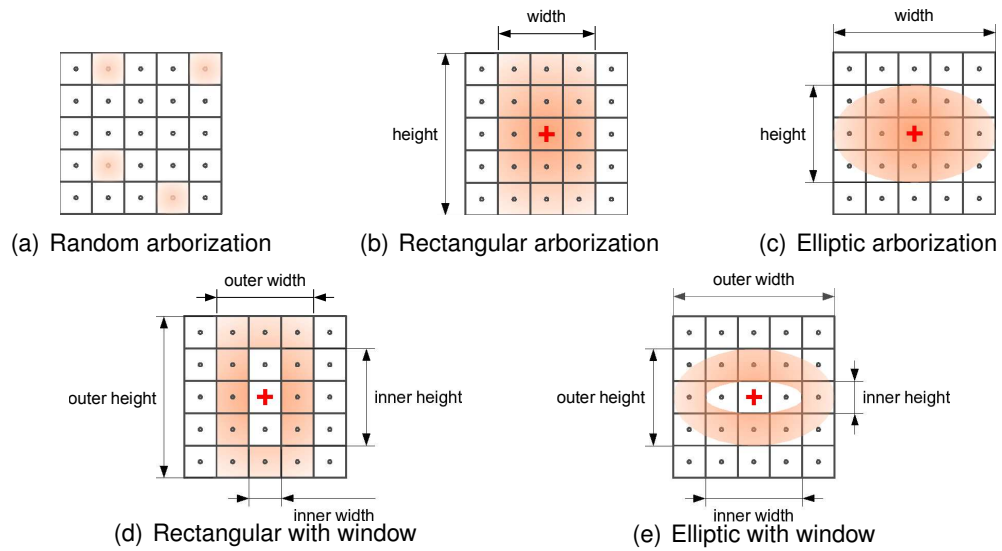


fig. 2.19: Arborization types

2.14.3 DelayFunction

The delay function is used to compute the delay for each synapse belonging to an arborization. In most delay functions the calculation is depending on the size of the arborization, i.e. height/width, or outer height/outer width respectively.

The possible delay functions are:

- *FunLinear*
- *FunGaussian*
- *FunBlock*
- *FunRandom*
- *FunUniform*

Figure 2.20 shows the meaning of the parameters for the three least trivial functions.

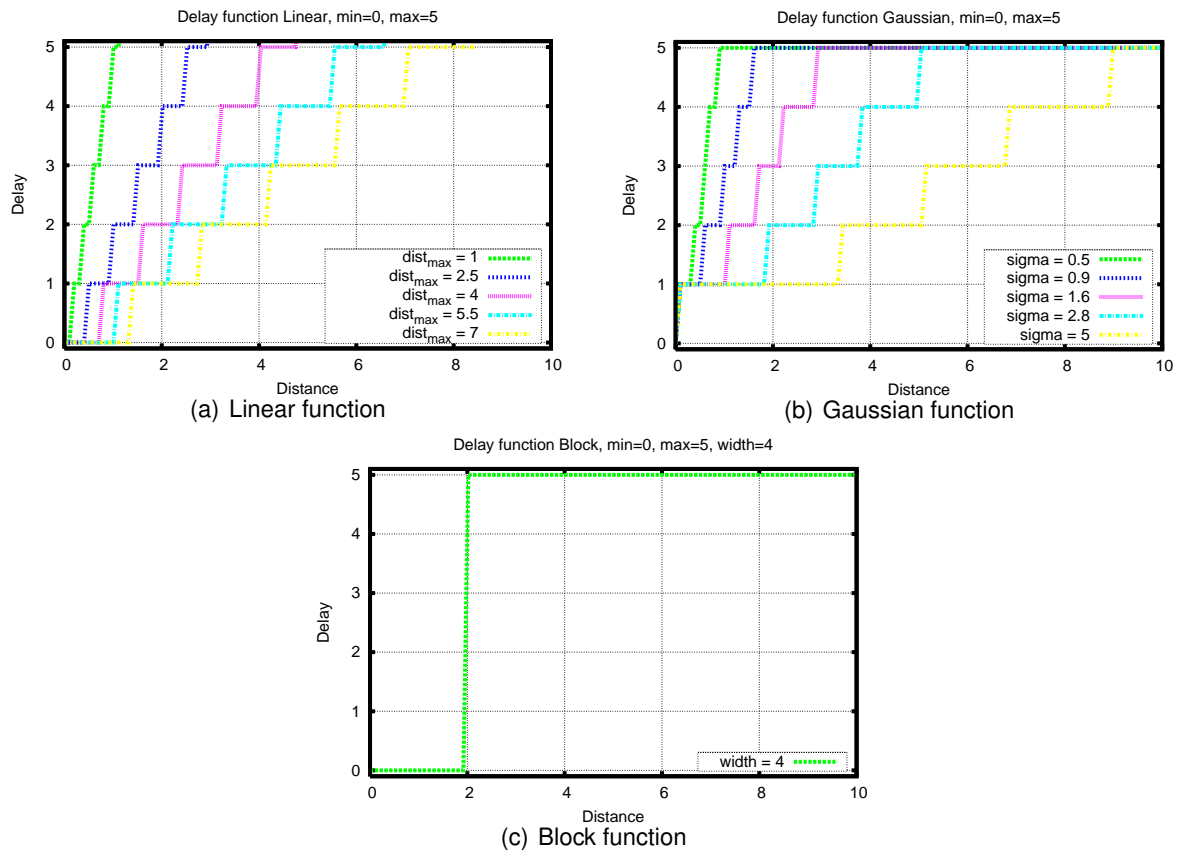


fig. 2.20: Delay functions

In *FunRandom* the delays are randomly distributed between 0 and *max*. Distance is not taken into account. The same holds true for *FunUniform*, where all synapses have the same delay of *value*.

2.14.4 AttenuationFunction

The attenuation function is used to compute the attenuation of the signal strength for each synapse belonging to an arborization. The larger the attenuation, the weaker the signal will be that arrives at the soma. In most attenuation functions the calculation is depending on the size of the arborization, i.e. height/width, or outer height/outer width respectively.

The types of attenuation functions are the same as for the delay (see above). Key difference between the delay and the attenuation function is that in the former case the values are continuous, whereas in the latter case they are discrete (fig. 2.21).

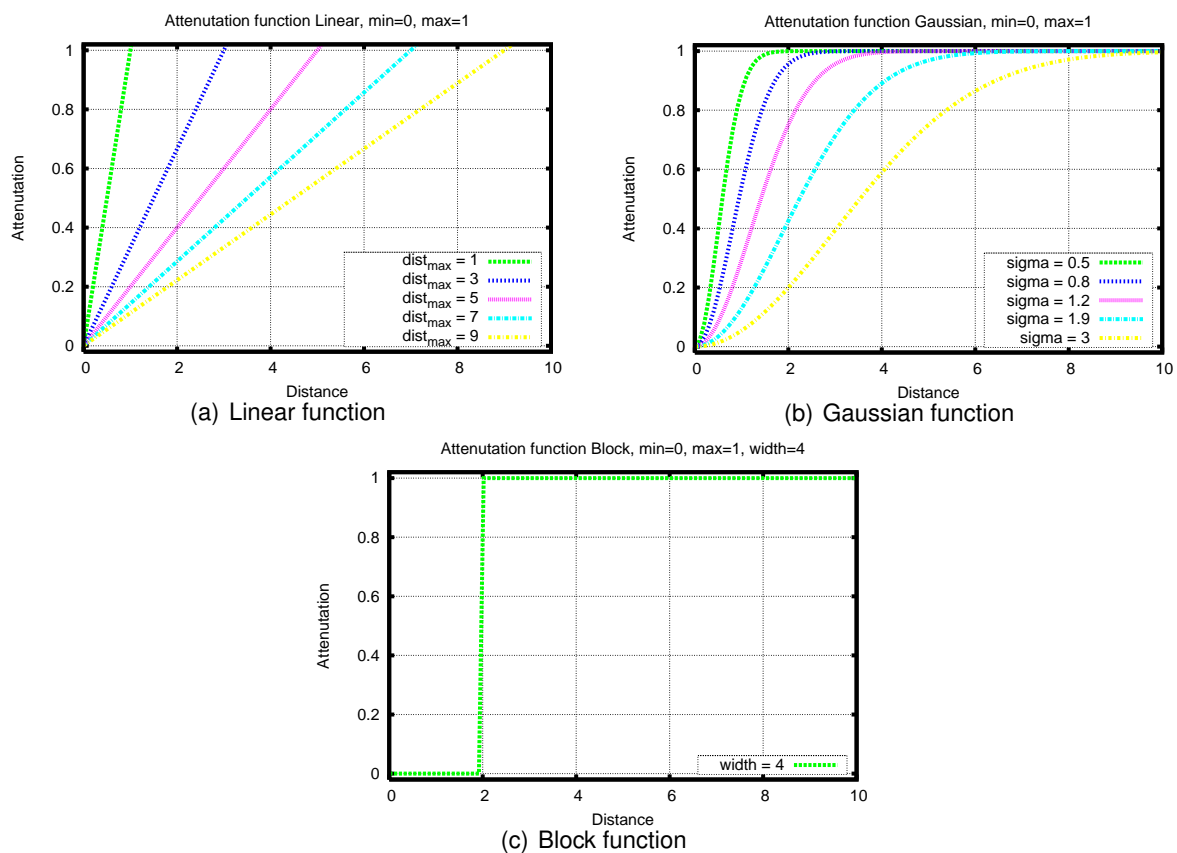



fig. 2.21: Attenuation functions

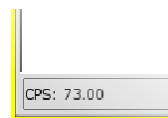
2.15 Running the simulation

To **start** the simulation, click on the play button  in the main toolbar (fig. 2.1⑦).



In the process of starting the simulation, `iqr` will check the model's consistency. If the model is not valid, e.g. because a connection target was not defined, a warning will pop up. You will not be able to start the simulation, unless the system is consistent.

While the simulation is running, the update speed in C(ycles) P(er) S(econd) is indicated in the lower left corner of the application window as shown in fig. 2.22.



To **stop** the simulation, click on the play button  again.

fig. 2.22:

2.16 Visualizing states and collecting data

This section introduces the facilities *iqr* provides to visualize and save the states of elements of the model. *Time plots* and *Group plots* are used to visualize the states of neurons, *Connection plots* serve to visualize the connectivity and the states of synapses of a connection. The *Data Sampler* allows to save data from the model to a file.

2.16.1 State Panel

iqr plots and the *Data Sampler* (section 2.16.6 on page 31) share part of their handling. Therefore, we will first have a look at some common features.

On the right side of the plots and the *Data Sampler*, you find a *State Panel* which looks similar to fig. 2.23. ① shows the name of the object from which the data originates. Neurons and synapses have a number of internal states that vary from type to type. In the frame entitled **states** you can select which state ② should be plotted or saved. Squares in front of the names indicate that several states can be selected simultaneously, circles mean that only one single state can be selected.

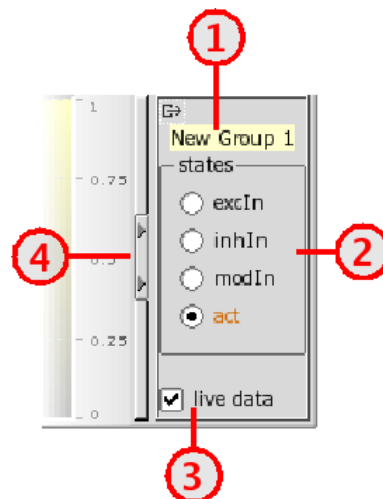


fig. 2.23:



In a newly created plot or *Data Sampler*, the check-box **live data** ③ will not be checked, which means that no data from this *State Panel* is plotted or saved. To activate the *State Panel*, check the tic by clicking the check-box.

To hide the panel with the states of the neuron or synapse, drag the bar ④ to the right side.


2.16.2 Drag & drop

The visualizing and data collecting facilities of *iqr* make extensive use of drag & drop.

You can drag groups from the *Browser* (© fig. 2.1, section 2.2 on page 8), the symbol in the top-left corner of *State Panels* (fig. 2.24①), or regions from *Space plots*.



fig. 2.24:

To **drag**, click on any of the above mentioned items, and move the mouse while keeping the button pressed. The cursor will change to . Move the mouse over the target and **drop** by releasing the mouse button.

The table below summarized what can be dragged where.

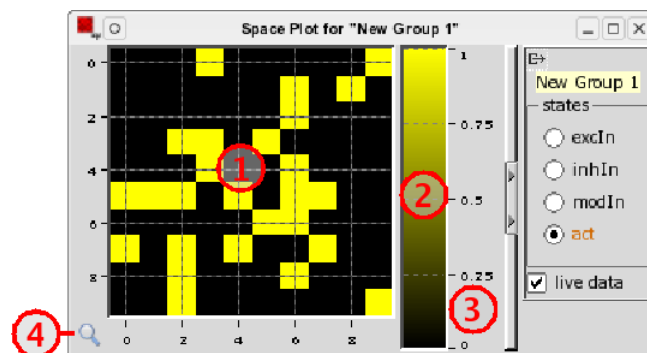
origin	target	what
<i>Browser</i>	→ <i>Time plot, Group plot, Data Sampler</i>	Group
<i>Time plot</i>	→ <i>Group plot, Data Sampler</i>	Group
<i>Data Sampler</i>	→ <i>Time plot, Group plot</i>	Group
<i>Group plot</i>	→ <i>Time plot, Data Sampler</i>	Group/Region

2.16.3 Space plots


A *Space plot* as shown in fig. 2.25 displays the state of each cell in a group in the plot area ①.

To create a new *Space plot*, right click on a group in the diagram editing pane or the browser and select **Space plot** from the context-menu.

The value for the selected state (see above) is color coded, the range indicated in the color bar ②. A *Space plot* solely plots one state of one group.

fig. 2.25: iqr *Space plot*

To change the mode of **scaling** for the color bar, right-click on the axis ③ to the right size. If **expand only** is ticked, the scale will only increase; if **auto scale** is selected, you can manually enter the value range.

You can **zoom** into the plot by first clicking  (fig. 2.25④), and selecting the region of interest by moving the cursor while keeping the mouse button pressed. To return to full-view, click into the plot area.

You can select a region of cells in the *Space plot* by clicking in the plot area (fig. 2.25①) and moving the mouse while holding the left button pressed. This region of cells can now be dragged and dropped onto a *Time plot*, *Space plot*, or *Data Sampler*.

To change the group associated to the *Space plot*, drop a group onto the plot area or the *State Panel*. Dropping a region of a group has the same

effect as dropping the entire group, as *Space plot* always plots the entire group.


2.16.4 Time plots

A *Time plot* (fig. 2.26) serves to plot the states of neurons against time.

To create a new *Time plot*, right click on a group in the diagram editing pane or the browser and select **Time plot** from the context-menu.

Time plots display the **average** value of the states of an entire group or region of a group. A *Time plot* can plot several states, and states from different groups at once. Each state is plotted as a separate trace.

To add a new group to the *Time plot* use drag & drop as described in section 2.16.2 on page 28, or drag and drop a region from a *Space plot* onto the plot area or one of the *State Panels*. If dropped onto the plot area, the group or region will be added, if dropped onto a *State Panel*, you have the choice to replace the *State Panel* under the drop point, or add the group/region to the plot.

To remove a group from the *Time plot* close the *State Panel* by clicking  in the top-right corner.

The *State Panels* for *Time plots* are showing which region of a group is plotted by means of a checker board (fig. 2.26②). Depending on the selection, the entire group or the region will be high-lighting in red.

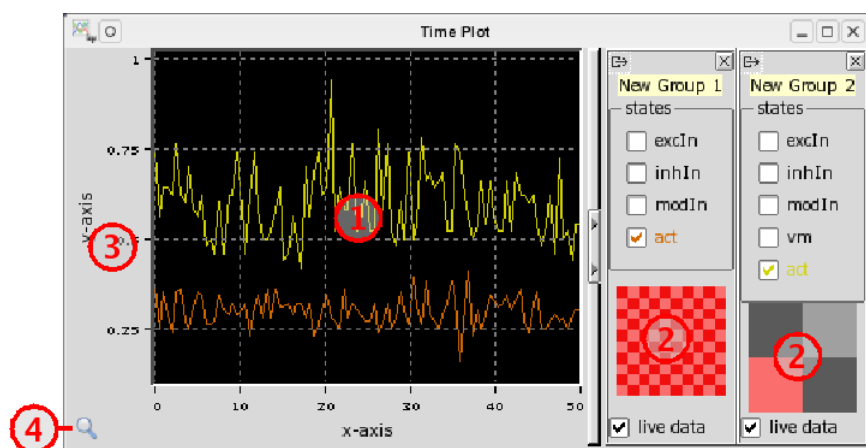



fig. 2.26: iqr *Time plot*

You can **zoom** into the plot by first clicking  (fig. 2.25④), and selecting the region of interest by moving the cursor while keeping the mouse button pressed. To return to full-view, click into the plot area.

To change the scaling of the y-axis, use the context-menu of the axis ③ (for detail see above, *Space plots*).

2.16.5 Connection plots

Connection plots (fig. 2.27) are used to visualize the static and dynamic properties of connections and synapses. The source group ① is on the left, the target group ② on the right side.

To create a new *Connection plot*, right click on a connection in the diagram editing pane, and select **Connection plot** from the context-menu.

The *State Panel* ③ for *Connection plots* is slightly different than for other plots. You can select to display the static properties of a connection, i.e. the **Distance** or **Delay**, or the internal states by selecting **Synapse**, and one of the states in the list.

To visualize the synapse value for the corresponding neuron, click on the cell in the source or the target group.



Connection plots do not support drag & drop.

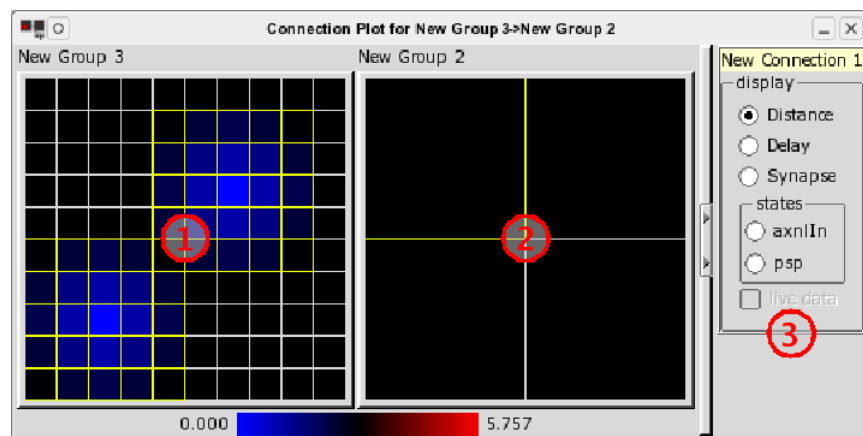


fig. 2.27: iqr *Connection plot*

2.16.6 Data Sampler

The *Data Sampler* (fig. 2.28) is used to save internal states of the model. To open the *Data Sampler* use the menu **Data** → **Data Sampler**.

A newly created *Data Sampler* is not bound to a group. To add groups or regions from groups use drag & drop as described earlier.

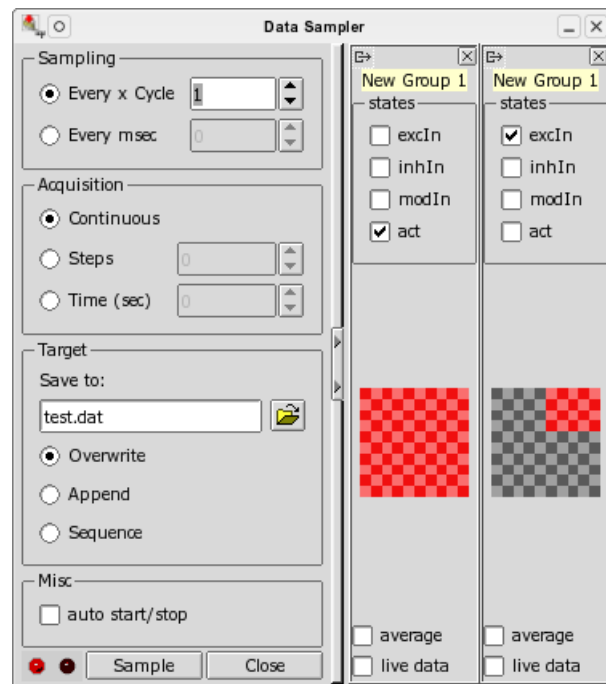
Like for the *Time plots*, the *State Panel* for the *Data Sampler* indicates which region of the group is selected. Moreover you will find an **average** check-box which allows you to save the average of the entire group or region of cells instead of the individual values.

The *Data Sampler* has the following options:

Sampling

Defines at what frequency data is saved.

- **Every x Cycle** Specifies how often data is saved in relation to the updates of the model, e.g. a value of 1 means data is saved

fig. 2.28: *Data Sampler*

at every update cycle of the model, a value of 10 that the data is saved only every 10th cycle.

Acquisition These options define for how long data is saved.

- **Continuous** Data is saved until you click on **Stop**.
- **Steps** Data is saved for as many update cycles as you define.

- Target**
- **Save to:** Name of the file where the data is saved.
 - **Overwrite** Overwrite the file at every start of sampling.
 - **Append** Append data to the file at every start of sampling.
 - **Sequence** Create a new file at every start of sampling. The name of the new file is composed of the base name from **Save to:** with a counter appended.

- Misc**
- **auto start/stop** Automatically start and stop sampling when the simulation is started and stopped.

If the **Sampling**, **Acquisition**, and **Target** options are set, and one or more groups/regions of cells are added, you can start sampling data. To do this, click on the **Sample** button. While data is being sampled you will see the lights next to the **Sample** button blinking. To stop sampling, click on the **Stop** button.



You will only be able to start sampling data if you specified the **Sampling**, **Acquisition**, and **Target** options, and one or more groups were added.

2.16.7 Saving and loading configurations

You can save the arrangement and the properties of the current set of plots and *Data Sampler* (section 2.16.6 on page 31). To do so, • go to **Data** → **Save Configuration** • select a location and file name.

To read an existing configuration file, select **Data** → **Open Configuration**.



If groups or connections were deleted, or sizes of groups changed between saving and loading the configuration, the effect of loading a configuration is undefined.

2.17 Manipulating states

The *State Manipulation* tool as shown in fig. 2.29 serves to change the activity (**Act**) of the neurons in a group. You can draw patterns, add them to a list, play them back with different parameters.

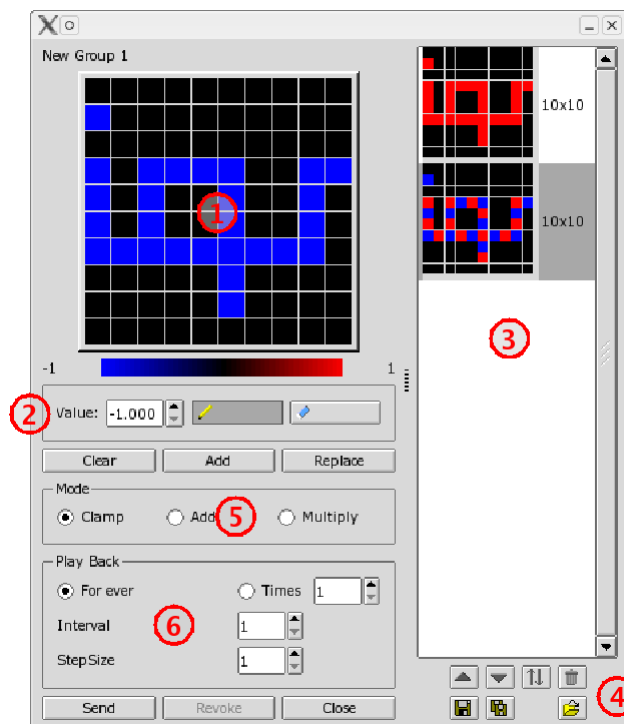


fig. 2.29: *State Manipulation* tool

Primer As the *State Manipulation* is a fairly complex tool, we will start with a step-by-step primer. The details are specified further below.



1. create a pattern by selecting  from ② to draw in the drawing area ①

2. add the pattern to the list ③

3. press **Send**



Only patterns in the pattern list are sent to the group.

Drawing a pattern To create a new pattern, • set the value using the **Value** spin-box in the ② drawing toolbar • select the , and draw the pattern in the drawing area ①. To clear the pattern drawing area, press **clear**, or to selectively erase points from pattern, press the  button.

You can now add the pattern to the pattern list ③ by clicking on **Add**, or you can replace the selected pattern in the list with your new pattern by pressing **Replace**.

Pattern list To manipulate the list of patterns, use the toolbar ④ below the list of patterns.



Move the selected pattern up or down in the list



Invert the order of the list



Delete the selected pattern



Save the list of pattern



Save the list of pattern under a new name



Open an existing list of pattern

Sending options As mentioned previously, the *State Manipulation* affects the activity of neurons. You can choose from three different **modes** ⑤:

Clamp The activity of the neurons is set to the values of the pattern

Add The values from the pattern are **added** to the activity of the neuron

Multiply The values from the pattern are **multiplied** to the activity of the neuron

In the frame **Play Back** ⑥, you can specify how many times the patterns in the list will be applied; either select **Forever** or **Times** and change the value in the spin-box next to it.

The **Interval** determines the number of time steps to wait prior to sending the next pattern to the group. E.g. a step size of 1 will send a pattern at each time step, step size 2 only at time step 1, 3, 5 etc.

StepSize controls which patterns from the list should be applied; 1 means every pattern, 2 means only the first, third, fifth etc. pattern.

Sending To send the patterns in the list to the group, press **Send**. If you have chosen to send the patterns **Forever**, **Revoke** will stop applying patterns to the group.

2.18 Support for work-flow of simulation experiments

Working with simulations employs a number of generic steps: Designing the system, running the simulation, visualising and analysing the behaviour of the model, perturbing the system, and tuning of parameters. Next to these steps, the automation of experiments and the documentation of the model form important parts of the work-flow. Subsequently we will describe the mechanisms *iqr* provides to support these tasks.

Central control of parameters and access from *Modules*: The *Harbor*

When running simulations, users frequently adjust the parameter of only a limited number of elements. Using the *Harbor*, users can collect system elements such as parameters of neurons and synapses in a central place (fig. 2.30), and change the parameters directly from the *Harbor*. A second function of the *Harbor* is to expose parameters to an *iqr Module*: All parameters collected in the *Harbor* can be queried and changed from within a *Module*. Using this method, parameter optimisation methods can be implemented directly inside *iqr*.

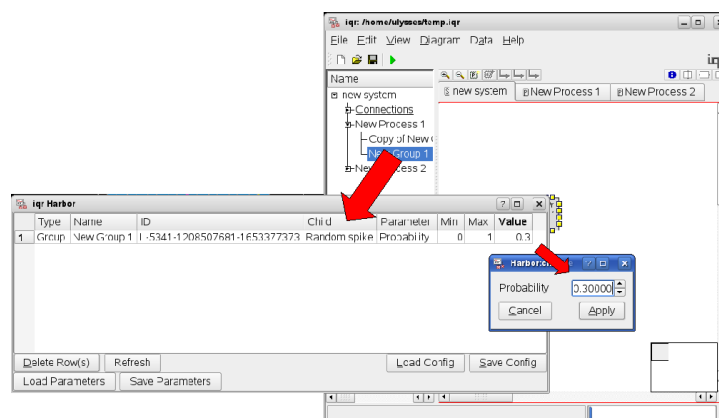


fig. 2.30: Using the *Harbor*, users can collect system elements such as parameters of neurons and synapses in a central place. Items are added to the *Harbor* by dragging them from the *Browser*. *Harbor* configurations can be saved and loaded.

Remote control of *iqr* In a number of use-cases, being able to control a simulation from outside the simulation software itself is useful. For this purpose, *iqr* is listening to incoming message on a user defined TCP/IP port. This allows to control the simulation and change parameters of the system. Concretely, this remote control interface supports the following syntax:

```
cmd:<COMMAND>
[;itemType:<TYPE>;
[itemName:<ITEM NAME>|itemID:<ITEM NAME>];
paramID:<ID>;value:<VALUE>;]
```

The supported **COMMAND** are: start, stop, quit, param, startsampler, stopsampler. The param command allows to change the parameter of

elements, and needs as an argument the type of item (ItemType: `PROCESS`, `GROUP`, `NEURON`, `CONNECTION`, `SYNAPSE`), the name (itemName) or ID (itemID) of the element, the ID of the parameter (ParamID), and the value to be set. Items can be addressed either by their name or their ID. In the case of name, all items with this name are changed. This feature can be used to change multiple elements at the same time.

Documentation of the system The documentation of a system comprises the descriptions of its static and dynamic properties. To document the structure of the model, *iqr* allows to export circuit diagrams in the `svg` and `png` image format or to print them directly. A second avenue of documenting the system is based on the Extensible Markup Language (XML) (<http://www.w3.org/XML/>) format of the system files in *iqr*. XML formatted text can be transformed into other text formats using the Extensible Stylesheet Language Family (XSL). One example of such an application is the transformation of a system file into the `dot` language for drawing directed graphs as hierarchies (<http://www.graphviz.org/>). Another transform is the creation of system descriptions in the \LaTeX typesetting system.

2.19 Modules

A *Module* in *iqr* is a plug-in to exchange data with an external entity. This entity can be a sensor (camera, microphone, etc.), an actor (robot, motor) or an algorithm.

- Modules read and write the activity of groups.
- A process can have only one single module associated to it.
- A module reads from and writes to groups that belong to the process associated with the module.

To assign a module to a group, open the process properties dialog (section 2.8 on page 13), select the module type from the pull-down menu of the **Module** frame, and press **Apply**.

If a process has a module associated to it, the process icon in the diagram will appear as in fig. 2.31.

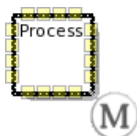


fig. 2.31:

Clicking on the **Edit** button in the **Module** frame will bring up the module property dialog. The parameters for the module vary from type to type and are explained in detail in Appendix Modules on page 45.

Most modules will feed data into and/or read from groups. Tabs **Groups to Module** and **Module to Groups** contain the allocation information between modules and groups. The pull-down menu next to the name of the reference will list all groups local to the current process. In the diagram you can identify a module's group input (blue arrow in the top right as shown in fig. 2.32) or groups with output to a module (blue arrow in the bottom right corner).



For reasons of system consistency, all references in the **Groups to Module** and **Module to Groups** tab must be specified.



You can disable the update of a module during the simulation by un-checking the **Enable Module** check-box in the process properties dialog.

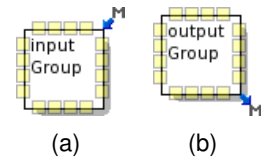


fig. 2.32:

3 Appendix

3.1 Neuron types

This section of the Appendix gives an overview of the neuron types most frequently used in `iqr`.

Input types Depending on the connection type, neurons in the post-synaptic group will receive either • excitatory (`excln`) • inhibitory (`inhln`) or • modulatory (`modln`) input.



Not all types of neurons make use of all three types of inputs.

3.1.1 Random spike

A random spike cell produces random spiking activity with a user-defined spiking probability. Unlike the other cell types, it receives no input and has no membrane potential. The output of a random spike cell i at time $t + 1$, $a_i(t + 1)$, is given by

$$a_i(t + 1) = \begin{cases} \text{SpikeAmpl} & \text{with probability Prob} \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

Parameters

name	Probability (Prob)
description	Probability of a spike occurring during a single time step
range	0.0 - 1.0

name	Spike amplitude (SpikeAmpl)
description	Amplitude of each spike
range	0.0 - 1.0

States

name	Activity (act)
description	

3.1.2 Linear threshold

Graded potential cells are modeled using linear threshold cells. The membrane potential of a linear threshold cell i at time $t + 1$, $v_i(t + 1)$, is given

by

$$\begin{aligned}
 v_i(t+1) = & \text{VmPrs}_i v_i(t) \\
 & + \text{ExcGain}_i \sum_{j=1}^m w_{ij} a_j(t - \delta_{ij}) \\
 & - \text{InhGain}_i \sum_{k=1}^n w_{ik} a_k(t - \delta_{ik}) \quad (3.2)
 \end{aligned}$$

where $\text{VmPrs}_i \in \{0, 1\}$ is the persistence of the membrane potential, ExcGain_i and InhGain_i are the gains of the excitatory (excln) and inhibitory (inhln) inputs respectively, m is the number of excitatory inputs, n is the number of inhibitory inputs, w_{ij} and w_{ik} are the strengths of the synaptic connections between cells i and j and i and k respectively, a_j and a_k are the output activities of cells j and k respectively, and $\delta_{ij} \geq 0$ and $\delta_{ik} \geq 0$ are the delays along the connections between cells i and j and cells i and k respectively. The values of w and δ are set by the synapse type and are described in Appendix Synapse types on page 44.

The output activity of cell i at time $t + 1$, $a_i(t + 1)$, is given by

$$a_i(t + 1) = \begin{cases} v_i(t + 1) & \text{with probability Prob for } v_i(t + 1) \geq \text{ThSet} \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

where ThSet is the membrane potential threshold, $\text{Rand} \in \{0, 1\}$ is a random number and Prob is the probability of activity.

Parameters

name	Excitatory gain (ExcGain)
description	Gain of excitatory inputs. The inputs are summed before being multiplied by this gain.
range	0.0 - 10.0
name	Inhibitory gain (InhGain)
description	Gain of inhibitory inputs. The inputs are summed before being multiplied by this gain.
range	0.0 - 10.0
name	Membrane persistence (VmPrs)
description	Proportion of the membrane potential remaining after one time step if no input arrives.
range	0.0 - 1.0
name	Clip potential (Clip)
description	Limits the membrane potential to values between VmMax and VmMin. Parameters: maximum potential, VmMax; minimum potential, VmMin
options	true, false

name	Minimum potential (VmMin)
description	Minimum value of the membrane potential
range	0.0 - 1.0
name	Maximum potential (VmMax)
description	Maximum value of the membrane potential
range	0.0 - 1.0
name	Probability (Prob)
description	Probability of output occurring during a single time step
range	0.0 - 1.0
name	Threshold potential (ThSet)
description	Membrane potential threshold for output activity
range	0.0 - 1.0

States

name	Membrane potential (vm)
name	Activity (act)

3.1.3 Integrate & fire

Spiking cells are modeled with an integrate-and-fire cell model. The membrane potential is calculated using equation (3.2). The output activity of an integrate-and-fire cell at time $t + 1$, $a_i(t + 1)$ is given by

$$a_i(t + 1) = \begin{cases} \text{SpikeAmpl} & \text{with probability Prob for } v_i(t + 1) \geq \text{ThSet} \\ 0 & \text{otherwise} \end{cases} \quad (3.4)$$

where SpikeAmpl is the height of the output spikes, ThSet is the membrane potential threshold, $\text{Rand} \in \{0, 1\}$ is a random number and Prob is the probability of activity.

After cell i produces a spike, the membrane potential is hyperpolarized such that

$$v'_i(t + 1) = v_i(t + 1) - \text{VmReset} \quad (3.5)$$

where $v'_i(t+1)$ is the membrane potential after hyperpolarization and VmReset is the amplitude of the hyperpolarization.

Parameters

name	Clip potential (Clip)
description	Limits the membrane potential to values between VmMax and VmMin. Parameters: maximum potential, VmMax; minimum potential, VmMin
options	true, false

name	Excitatory gain (ExcGain)
description	Gain of excitatory inputs. The inputs are summed before being multiplied by this gain.
range	0.0 - 10.0

name	Inhibitory gain (InhGain)
description	Gain of inhibitory inputs. The inputs are summed before being multiplied by this gain.
range	0.0 - 10.0

name	Membrane persistence (VmPrs)
description	Proportion of the membrane potential remaining after one time step if no input arrives.
range	0.0 - 1.0

name	Minimum potential (VmMin)
description	Minimum value of the membrane potential
range	0.0 - 1.0

name	Maximum potential (VmMax)
description	Maximum value of the membrane potential
range	0.0 - 1.0

name	Probability (Prob)
description	Probability of output occurring during a single time step
range	0.0 - 1.0

name	Threshold potential (ThSet)
description	Membrane potential threshold for output of a spike
range	0.0 - 1.0

name	Spike amplitude (SpikeAmpl)
description	Amplitude of output spikes
range	1.0 - 1.0

name	Membrane potential reset (VmReset)
description	Membrane potential reduction after a spike
range	0.0 - 1.0

States

name	Activity
direction	output

3.1.4 Sigmoid

The `iqr` sigmoid cell type is based on the perception cell model often used in neural networks. The membrane potential of a sigmoid cell i at time $t + 1$, $v_i(t + 1)$, is given by equation (3.2). The output activity, $a_i(t + 1)$ is given by

$$a_i(t + 1) = 0.5 * (1 + \tanh(2 * Slope * (v_i(t + 1) - ThSet))) \quad (3.6)$$

where *Slope* is the slope and *ThSet* is the midpoint of the sigmoid function respectively.

Parameters

name	Clip potential (Clip)
description	Limits the membrane potential to values between VmMax and VmMin. Parameters: maximum potential, VmMax; minimum potential, VmMin
options	true, false
name	Excitatory gain (ExcGain)
description	Gain of excitatory inputs. The inputs are summed before being multiplied by this gain.
range	0.0 - 10.0
name	Inhibitory gain (InhGain)
description	Gain of inhibitory inputs. The inputs are summed before being multiplied by this gain.
range	0.0 - 10.0
name	Membrane persistence (VmPrs)
description	Proportion of the membrane potential remaining after one time step if no input arrives.
range	0.0 - 1.0
name	Minimum potential (VmMin)
description	Minimum value of the membrane potential
range	0.0 - 1.0
name	Maximum potential (VmMax)
description	Maximum value of the membrane potential
range	0.0 - 1.0
name	Midpoint
description	Midpoint of the sigmoid
range	0.0 - 1.0

name	Slope
description	Slope of the sigmoid
range	0.0 - 1.0

States

name	Activity (act)
name	Membrane potential (vm)

3.2 Synapse types

Apical shunt

States

name
direction
description

Postsynaptic potential

output

Feedback

name
description

Apical dendrite backpropogation

name
description

Apical dendrite shunt

Fixed weight

Parameters

name
description
range/options

Weight

Uniform synaptic weight.
-1.0 - 1.0

States

name
direction
description

Postsynaptic potential

output

Uniform fixed weight

Parameters

name
description
range/options

Weight

Uniform weight for all synapses.
-1.0 - 1.0

States

name
direction
description

Postsynaptic potential

output

3.3 Modules

In this appendix, the various modules, their application and parameters are presented. The list below gives an overview of the standard modules that come with `iqr`.

Additional modules can be found at:

<http://sourceforge.net/projects/iqr-extension/>

3.3.1 Threading

Some of the modules are running in their own thread. This means that their update speed is independent of the update speed of the main simulation. All modules relating to robots are threaded, whereas video modules are not threaded. In some cases a low update speed of the video hardware slows down the entire simulation. Therefore make sure to employ all means (like compression) to get a good update speed for the hardware.

3.3.2 Robots

Khepera and e-puck

Khepera: **Operating system:** Linux

e-puck: **Operating system:** Linux, Windows

This chapter describes the `iqr` interfaces to the Khepera® (K-Team S.A. Lausanne, Switzerland, <http://www.k-team.com/>) and e-puck mobile robot (<http://www.e-puck.org/>). More information about the robots can be found in the e-puck Mini Doc (<http://www.gctronic.com/files/miniDocWeb.pdf>) and the Khepera User Manual (<http://ftp.k-team.com/khepera/documentation/KheperaUserManual.pdf>).

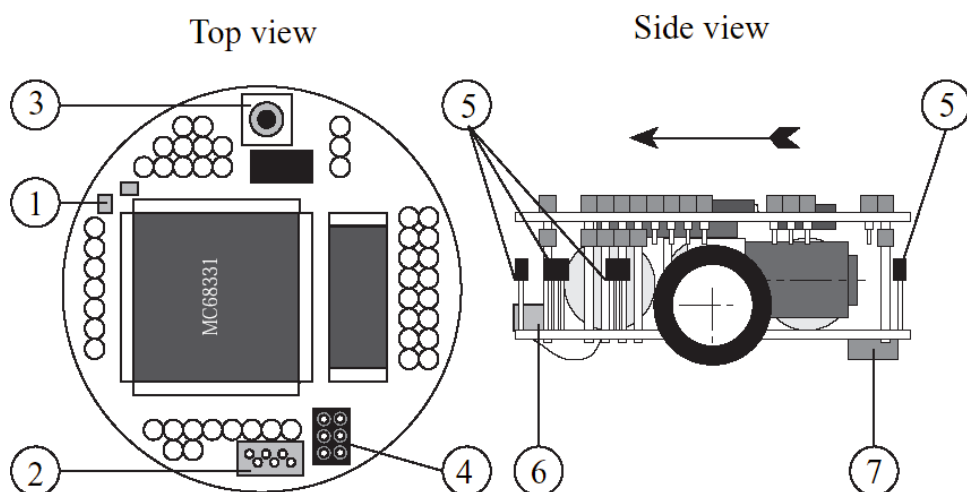


fig. 3.1: Schematic drawing for the Khepera robot (from Khepera User Manual)
 1. LEDs, 2. Serial line connector, 3. Reset button, 4. Jumpers for the running mode selection, 5. Infra-Red sensors, 6. Battery recharge connector, 7. ON - OFF battery switch, 8. Second reset button

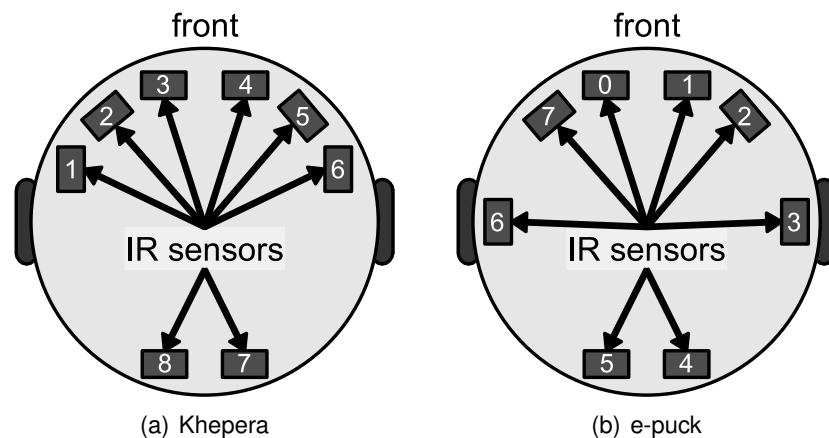


fig. 3.2: Arrangement of infra-red sensors

A full description of the features of the robots can be found in the respective manuals, here only the most important features are described.

Communication The communication with the computer is done over a serial cable (Khepera) or via BlueTooth (E-puck).

Parameters

name	Serial Port
description	Path to Serial Port. Under Linux this will be for example if connected via a cable <code>"/dev/tty0"</code> or <code>"/dev/rfcomm0"</code> if connected via BlueTooth. Under Windows this will be for example <code>"COM20"</code>
default	Khepera: <code>/dev/ttyS0</code> . E-puck: <code>/dev/rfcomm0</code>
name	Show output
description	Show Khepera Display
options	true, false
name	Frontal LED on
description	Turn Frontal LED on
options	true, false
name	Lateral LED on
description	Turn Lateral LED on
options	true, false

Control . Parameters

name	Max Speed (MaxSpeed)
description	Maximum speed for the robot
range	0.0 - 1000.
name	Type of Motor-Map
description	See below and fig. 3.3a and 3.3b
options	<i>VectorMotorMap</i> or <i>TabularMotorMap</i>
name	Speed Controller Proportional
description	Set Speed Controller Proportional Value
range	0 - 3800
name	Speed Controller Integral
description	Set Speed Controller Integral Value
range	0 - 3800
name	Speed Controller Differential
description	Set Speed Controller Differential Value
range	0 - 3800

States

name	Proximal Sensors Output
direction	Module → Group
description	
size	9
range	0.0 - 1.0
name	Ambient Sensors Output
direction	Module → Group
description	
size	9
range	0.0 - 1.0
name	Position Monitor Output
direction	Module → Group
description	Two cells that can be used to monitor the x and y position of the robot.
size	2
range	0.0 - 1.0
name	Motor Input
direction	Group → Module
description	Motor command for the robot
size	9 * 9 for <i>TabularMotorMap</i> , 2*2 for <i>VectorMotorMap</i>
range	0.0 - 1.0

Tabular vs. Vector Motor-Map There are two ways to control the motors of the robot. In *Tabular* motor-map, the cell with the highest activity in the

Motor Input group is taken, and from the position of this cell in the lattice of the group, the movement of the robot is computed. Figure 3.3a) shows the Tabular motor-map. For each cell, the corresponding direction and the speed of the left (red) and right (blue) motor is shown.

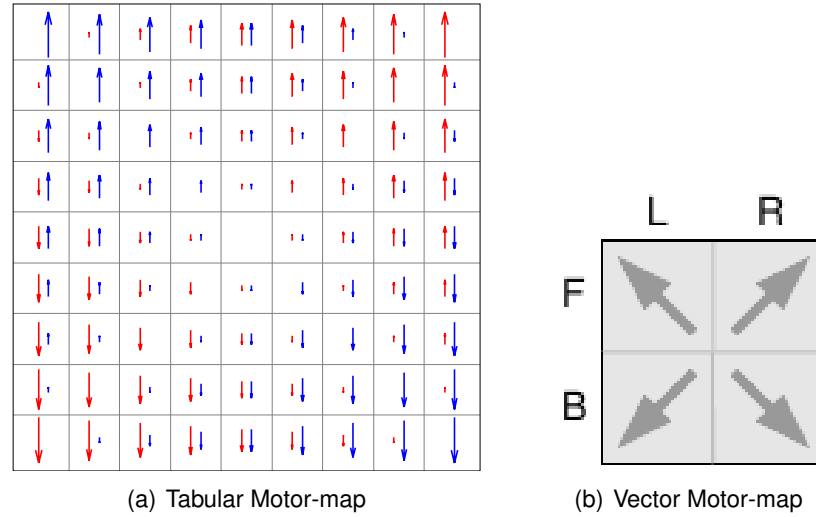


fig. 3.3: Moto-map types used for the Khepera and e-puck robots.

In *Vector* motor-map (fig. 3.3b)) vector for the direction and speed of motion is computed from the activity of all cells in the **Motor Input** group.

$$Motor_{left} = (Act_{(1,1)} - Act_{(1,2)}) * MaxSpeed/2$$

$$Motor_{right} = (Act_{(2,1)} - Act_{(2,2)}) * MaxSpeed/2$$

Infra-red sensors KRobots are equipped with infra-red sensors. These sensors are used in two ways: • in passive mode, they function as light sensors • in active mode, they emit infra-red light and measure the amount of reflected light, which allows the robot to detect nearby objects. In fig. 3.4 the characteristics of the passive and active mode of the sensors is shown. Output of the passive sensing is fed into the **Ambient Sensors** cell group, active mode sensor readings into the **Proximal Sensors** cell group.

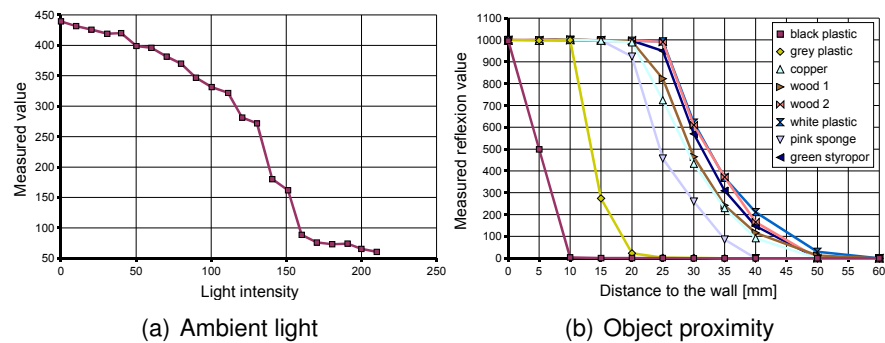


fig. 3.4: Khepera Infra-red sensors characteristics

3.3.3 Video

Operating system: Linux, Windows

The below the most important parameters of the video module are listed. Some parameters are only available under Linux.



Linux: If you encounter problems with the video module, run the application “xawtv” to check whether video capturing works for your computer.

Parameters

name	Video Device (Linux Only)
description	Path to Video Device. If the computer has more than one capture card, or an USB video device is attached in addition to an existing capture card, you will have to change to <i>/dev/video1</i> or higher.
default	<i>/dev/video0</i>

name	Show output
description	Show camera output
options	true, false

name	Image Width (Linux Only)
description	Width of the image to grab
range/options	160 - 640

name	Image Height (Linux Only)
description	Height of the image to grab
range/options	120 - 480

name	HSV
description	This option allows to choose between the hue, value, saturation (HSV), and red, green, blue (RGB) color spaces
options	true, false

name	Monochrome
description	If set to true, one a monochrome image will be acquired. The values will be written into the "Red/Hue" group.
options	true, false

Output States

name **Video Output Red/Hue/Grey**
 direction Module → Group
 range 0.0 - 1.0

name **Video Output Green/Saturation**
 direction Module → Group
 range 0.0 - 1.0

name **Video Output Blue/Value**
 direction Module → Group
 range 0.0 - 1.0



The sizes of the three color groups have to be the same.

3.3.4 Lego MindStorm**Operating system:** Linux

This module provides an interface to the Robotics Invention System 1.0 from Lego® MindStorms™. It is used for online control of the motors and sensors.

The hardware consists of a serial infra-red transmitter, the RCX microcomputer brick, motors, and sensors for touch, light, and rotation. The serial infra-red transmitter is connected to the serial port of the computer. The RCX microcomputer brick receives information from the transmitter and interfaces to motors and sensors (fig. 3.5).

Parameters

name **SensorMode**
 description Mode for each sensor
 options *set, raw, boolean*

name **SensorType**
 description Type for each sensor
 options *raw, touch, temp, light, rot*

name **Serial Port**
 description Path to Serial Port
 default */dev/ttyS0*

States

name **Sensors**
 direction Module → Group
 description Sensor readings
 size 3

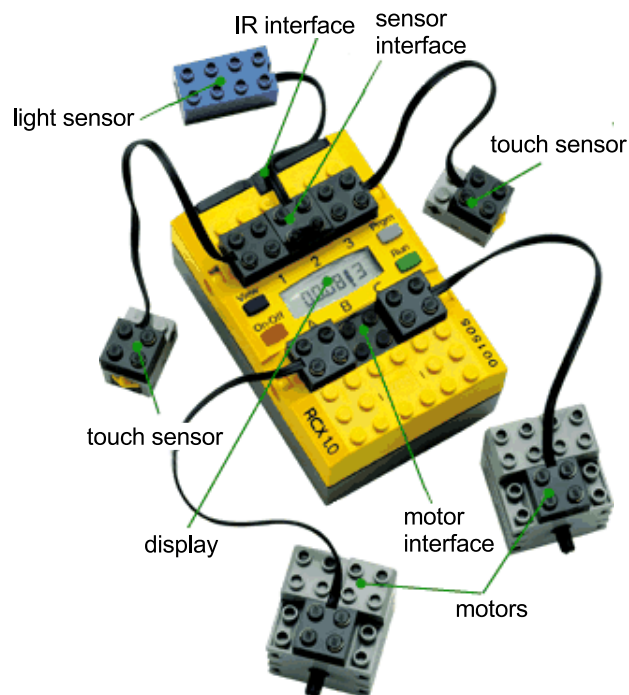


fig. 3.5: RCX Interface brick for Lego MindStorm (© LEGO Company)

name	Motor Input
direction	Group → Module
description	Motor power. The used range is $\geq 0, \leq 1$
size	6 (even number required)

name	Float Input
direction	Group → Module
description	A value > 0 sets the motor to zero resistance mode
size	3

name	Flip Input
direction	Group → Module
description	A value > 0 changes the direction of the motor
size	3

3.3.5 Serial VISCA Pan-Tilt

Operating system: Linux

This module can control VISCA based pan-tilt cameras such as the Sony® EVID100. A Pan-tilt device is a two-axis steerable mounting for a camera. Pan is the rotation in the horizontal, tilt the rotation in the vertical plane (fig. 3.6a)).

Parameters

name	Camera ID
description	ID of the camera (can be set on the camera itself)
default	1
name	Pan/Tilt Device
description	Path to Serial Port.
default	/dev/ttyS0
name	Pan/Tilt Mode
description	Pant-Tilt Mode
range/options	relative or absolute
name	Nr. Steps Pan
description	Number of steps for pan in relative mode
range	1 - 1000
name	Nr. Steps Tilt
description	Number of steps for tilt in relative mode
range	1 - 1000

States	
name	Pan/Tilt Input
direction	Group → Module
description	In <i>relative</i> Pan/Tilt
size	2 * 2

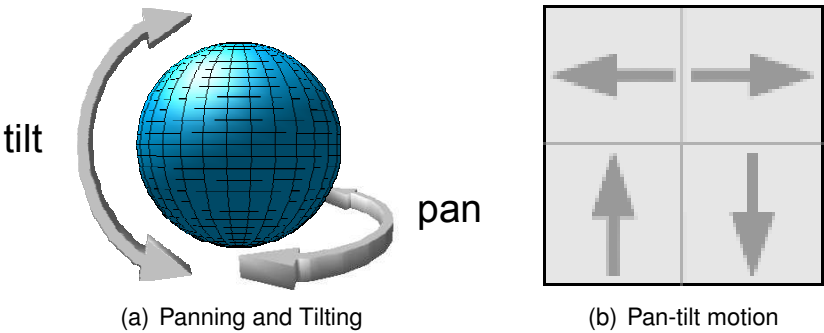


fig. 3.6:

Pan/Tilt Mode In fig. 3.6b), the association between the cells in the **Pan/Tilt Input** group and the motion of the pan-tilt device is shown.

In *absolute* mode, the activity of the cells in the **Pan/Tilt Input** group, defines the absolute position of the device:

$$Pan = Act_{(1,1)} - Act_{(2,1)}$$
$$Tilt = Act_{(1,2)} - Act_{(2,2)}$$

If the mode is set to *relative* the pan and the tilt angles are increased, based

one the cell's activity in the **Pan/Tilt Input** group:

$$\Delta_{pan} = (Act_{(2,1)} - Act_{(1,1)}) * PanFactor$$

$$\Delta_{tilt} = (Act_{(1,2)} - Act_{(2,2)}) * TiltFactor$$

where

$$PanFactor = (Pan_{max} - Pan_{min}) / nrsP$$

$$TiltFactor = (Tilt_{max} - Tilt_{min}) / nrsT$$