

iqr: A Tool for the Construction of Multi-level Simulations of Brain and Behaviour

Ulysses Bernardet · Paul F. M. J. Verschure

Published online: 26 May 2010
© Springer Science+Business Media, LLC 2010

Abstract The brain is the most complex system we know of. Despite the wealth of data available in neuroscience, our understanding of this system is still very limited. Here we argue that an essential component in our arsenal of methods to advance our understanding of the brain is the construction of artificial brain-like systems. In this way we can encompass the multi-level organisation of the brain and its role in the context of the complete embodied real-world and real-time perceiving and behaving system. Hence, on the one hand, we must be able to develop and validate theories of brains as closing the loop between perception and action, and on the other hand as interacting with the real world. Evidence is growing that one of the sources of the computational power of neuronal systems lies in the massive and specific connectivity, rather than the complexity of single elements. To meet these challenges—multiple levels of organisation, sophisticated connectivity, and the interaction of neuronal models with the real-world—we have developed a multi-level neuronal simulation environment, *iqr*. This framework deals with these requirements by directly transforming them into the core elements of the simulation

environment itself. *iqr* provides a means to design complex neuronal models graphically, and to visualise and analyse their properties on-line. In *iqr* connectivity is defined in a flexible, yet compact way, and simulations run at a high speed, which allows the control of real-world devices—robots in the broader sense—in real-time. The architecture of *iqr* is modular, providing the possibility to write new neuron, and synapse types, and custom interfaces to other hardware systems. The code of *iqr* is publicly accessible under the GNU General Public License (GPL). *iqr* has been in use since 1996 and has been the core tool for a large number of studies ranging from detailed models of neuronal systems like the cerebral cortex, and the cerebellum, to robot based models of perception, cognition and action to large-scale real-world systems. In addition, *iqr* has been widely used over many years to introduce students to neuronal simulation and neuromorphic control. In this paper we outline the conceptual and methodological background of *iqr* and its design philosophy. Thereafter we present *iqr*'s main features and computational properties. Finally, we describe a number of projects using *iqr*, singling out how *iqr* is used for building a “synthetic insect”.

Keywords Neuronal simulation · Multi-level · Large-scale · Synthetic epistemology · Bio-robotics

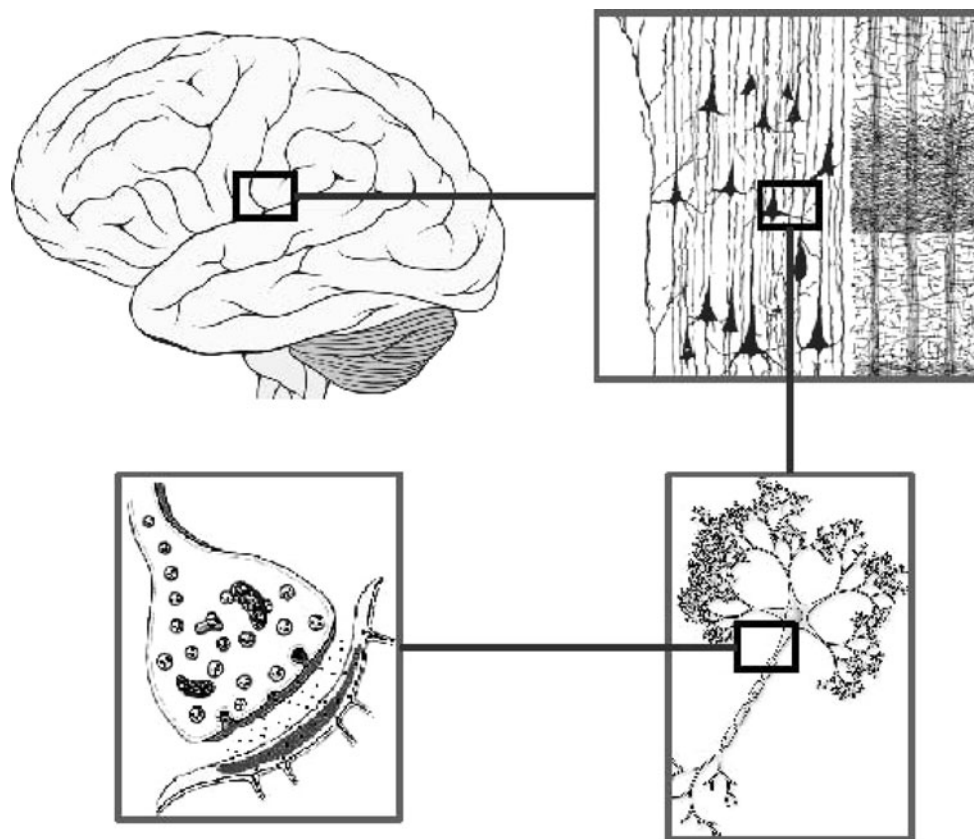
U. Bernardet (✉) · P. F. M. J. Verschure
Laboratory of Synthetic Perceptive, Emotive,
and Cognitive Systems (SPECS), Universitat Pompeu Fabra,
Roc Boronat 138, 08018 Barcelona, Spain
e-mail: ulysses.bernardet@upf.edu

P. F. M. J. Verschure
Catalan Institute of Research and Advanced Studies
(ICREA), Barcelona, Spain
e-mail: paul.verschure@upf.edu

Introduction

The brain is an extraordinarily complex machine. The workings of this machine can be described at a multitude of levels of abstractions and in various description languages (Fig. 1). These description levels range

Fig. 1 The levels of organisation of the brain range from neuronal sub-structures to circuits to brain areas



from the study of the genome and the use of genes in genomics; the investigation of proteins in proteomics; the detailed models of neuronal sub-structures like membranes and synapses in compartmental models; the networks of simple point neurons; the assignment of function to brain areas using e.g. brain imaging techniques; the mapping of input to output states as in psychophysics and the abstraction of symbol manipulation.

Level of Abstraction The different levels of abstraction are not mutually exclusive, but must be combined into a multi-level description. Focusing on a single level of abstraction can fall short where only a holistic, systemic view can adequately explain the system under investigation. One such example can be found in the phenomenon of behavioural feedback which indicates that behaviour itself can induce neuronal organisation (Verschure et al. 2003).

The Synthetic Approach We argue that an essential tool in our arsenal of methods to advance our understanding of the brain is the construction of artificial brain-like systems. The maxim of the synthetic approach—“truth and the made are convertible”—was put forward by the 17th century philosopher Giambattista Vico (1976). The key argument is that the

structure and the parameters of man-made, synthetic product are fostering our understanding of the modelled system. Yet Vico’s proposition brings about two other important aspects. Firstly, it is the process of building as such that is yielding new insights; in building we are compelled to explicitly state the target function of the system (and herein possibly err). Since we build all elements of the system to fulfil certain functions, we explicitly assign meaning to all elements and relations between elements of the system, which implies an understanding of the role of the elements and their interactions within the system to achieve its goal. Moreover, construction entails that we make explicit statements about the abstractions we make. Secondly, man-made devices are open to unlimited measurement and manipulation. Synthetic systems provide unlimited access to all internal states of the system, and can be manipulated at all levels, as well as at the local and global scale.

Large-scale Models If we want to build systems able to generate a meaningful behaviour in the real world, they have to be complete in the sense that they must span from sensory processing to the behavioural output. Systems compliant with this requirement will inevitably be of a large-scale, where the overall architecture is of critical importance.

Connectivity A human brain consists of about 100 billion neurons. Neurons, in turn, are vastly outnumbered by synapses; the estimations of the numbers of synapses per average neuron ranging from 1,000 to 10,000 (Shepherd 2003). This impressive ratio of number of neurons vs. number of synapses speaks in favour of the view that biological neuronal networks draw their computational power from the large-scale integration of information in the connectivity between neurons.

Real-world Bio-robotics The usage of robots in cognitive sciences has been heralded at the start of the 20th century by Hull and Tolman. Hull undertook the construction of the “psychic” machine (Hull 1952; Hull and Baernstein 1929; Baernstein and Hull 1931; Hull and Krueger 1931) by applying a framework of physics to psychology. Tolman, exploring an alternative conceptual route and striving at uniting the methods of behaviourism with the concepts of Gestalt psychology, in 1939 proposed the robot “Sowbug” (Tolman 1939). The shortcoming of many of the above mentioned approaches is that they are lacking a clear conceptualisation of the employment of robots in biological sciences. We see the main gain of using robots in the fact that the “real world” provides clear constraints at both, the construction and validation level: The properties of the elements of the model have to obey the laws of physics in their construction as well as in the interaction with the real world. If the agent is to operate in the real world, the mechanical properties have to take into account inertia, friction, gravity, energy consumption etc. Moreover, acquiring information from the environment will have a limited bandwidth and the data will most likely be noisy. One of the key properties of bio-robotics approach is that it circumvents the problem of the low degree of generalisability of models developed in simulation only. This problem stems from the limited input space to the model and the hidden a priori assumptions about the environment in which the system is to behave.

Enter *iqr*

To meet the challenges outlined so far—taking into account multiple levels of abstraction, large-scale models, complex connectivity, and the interaction of neuronal models with the real-world—we have developed a multi-level neuronal simulation environment, *iqr*, that exactly tries to deal with these open issues.

The dynamics of neuronal systems can be described best in terms of signal processing and flow. This paradigm lends itself for a graphical approach, allowing

to concentrate on the design of the data flow without the need to deal with implementational details, and to effectively explore the organisation of neuronal systems. Moreover, a graphical interface yields a low threshold for the use of the tool and a flat learning curve. In the domain of test, measurement, and control applications this approach has been successfully pursued for the past 20 years by LabVIEW™ (<http://www.ni.com/>). *iqr* implements this paradigm, allowing the user to perform most of the above mentioned steps in working with neuronal simulations using a graphical interface (Fig. 4).

iqr provides an efficient graphical environment yielding flexible user interaction to design large-scale multi-level neuronal systems that are scalable and extensible. The ability to interface the neuronal simulations with real-world devices—robots in the broader sense—in real-time is one of the cornerstone features. As a simulation tool, *iqr* fits in between high-level, general purpose simulation tools such as Matlab® (<http://www.mathworks.com/>) and highly specific, low-level, neuronal systems simulators such as NEURON (Hines and Carnevale 1997). The advantages over a general purpose tool are on the one hand the pre-defined 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, attention to detail is favoured over size of the model and speed of the simulation.

The 2006 workshop report of the OECD’s international neuroinformatics coordinating facility (INCF) (Djurfeldt and Lansner 2006) lists a number of important properties that simulator software should have. These include: availability of the source code, scalability of the model size, framework to implement user-defined extensions, support for different operating systems, facilitation of the exchange of models and data, a graphical interface, and documentation of the software.

Today a wide range of neuronal simulators is available (for a detailed review of major packages see Brette et al. (2007)). Many of these software packages, however, do not allow the user to design the system graphically and have no intrinsic mechanism to control robots or interface the other I/O devices. For a number of software packages, access to the source code is not granted, though access to the source code of the software is vital to ensure quality control and reproducibility of results (Amari et al. 2003). For this reason, we have released

the sources of *iqr* under the GNU General Public License (GPL).

The current version of *iqr* is a complete redesign and reimplement of the original version (Verschure 1997). *iqr* is implemented in C++ using the Qt widget-set (<http://www.trolltech.com/>), and runs on the Linux, Apple's Mac OS X, and Microsoft Windows platform. *iqr* is fully documented, including tutorials and an introduction on how to write user-specific extension. The sources and binaries (RPM and DEB for Linux, installer for Windows) of *iqr* can be downloaded from <http://iqr.sourceforge.net/>. From this page a bootable Linux Live system based on the Fedora Project Live CD (<http://fedoraproject.org/wiki/FedoraLiveCD>) can be downloaded. This allows to run *iqr* without the need for any installation. Additionally, developmental packages for writing user-defined extensions (*iqr-devel*) and a collection of tools (*iqr-tools*) are available.

From March 2007 to November 2009 the *iqr* web site on sourceforge.net was accessed 115,142 times (Median 2694), with a total of 3,202 (Median 74) downloads. *iqr* has been disseminated through a number of teaching and workshop activities, including the Telluride Neuromorphic Engineering Workshop 2004, and the Barcelona Cognition Brain and Technology Summer School (BCBT) 2008, 2009. Since 2007 *iqr* is used to introduce students to methods of neuronal modelling and system construction in the Interdisciplinary Master in Cognitive Systems and Interactive Media (CSIM) at the Universitat Pompeu Fabra, Barcelona, Spain. Additionally, *iqr* is used in a number of European projects such as "Synthetic Forager" (FP7 ICT project

no. 217148), NeuroChem (FP7 ICT project no. 216916), and ReNaChip (FP7 ICT project no. 216809).

In the subsequent part of the article we will show how *iqr* fulfils the aforementioned criteria, describe the design philosophy behind *iqr*, demonstrate how *iqr* is used to develop multi-level neuro-ethological models, and give examples of projects where *iqr* is used.

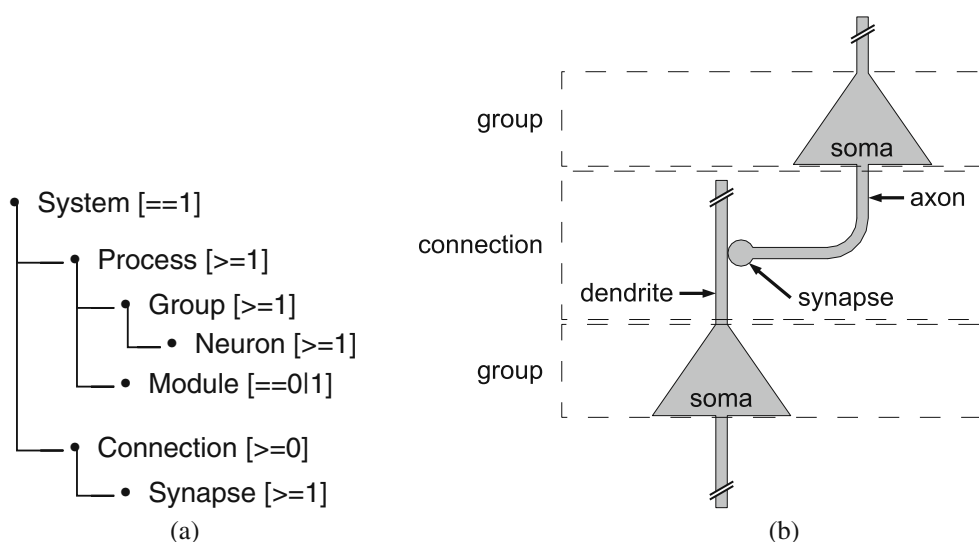
Developing Neuronal Systems with *iqr*

Functional Organisation of an *iqr* Model

The brain is organised at many different levels: areas, circuits, neurons (Fig. 1). *iqr* reflects this hierarchical organisation in the architecture of its systems. Models in *iqr* are organised at three different levels (Fig. 2a): The top or **system** level comprises of an arbitrary number of **processes**, and **connections**. Processes, in turn, consist of an arbitrary number of **groups**, and provide the means to structure the model into logical units as well as to interface it to external devices. A group is an aggregation of neurons of identical type. Connections are used to send signals from group to group, and are made up of synapses of identical type, plus the definition of the arborisation pattern of the dendrites and axons (Fig. 2).

To illustrate the structure of *iqr* systems at the neuronal level, it is useful to look at a simple system comprising of two groups with one connection between them (Fig. 2b). Within *iqr* groups and connections are conceived as two different frameworks. Groups comprise of the **somas** of neurons, where all inputs

Fig. 2 a The structure of models in *iqr*, numbers in brackets indicate how many elements are possible at what level. **b** The group and connection framework. Groups comprise of the **somas** of neurons, where all inputs are integrated. **Axon**, **synapse**, and **dendrite** are part of the connection framework



are integrated using the function of the specific neuron type, while the connection framework deals with the **axon**, **synapse**, and **dendrite** of the neurons. A connection includes the definition of the update function of the synapse, and the definition of the connectivity, i.e. the spatial layout of axons, dendrites and their interconnectivity. The synapses support the integration of all axonal and back-propagating dendritic inputs and compute the overall output according to their specific update function. Every synapse has a local “memory” and thus can adapt parameters of its update function, e.g. the synaptic strength, based on its individual history. A single **connection** between two neurons is an assembly of axon-synapse-dendrite bundles or **nexuses** (Fig. 3). A single nexus can connect several pre-synaptic neurons to **one** post-synaptic cell, or feed information from **one** pre-synaptic to multiple post-synaptic neurons. A nexus can hence comprise several axons, synapses, and dendrites (Fig. 3).

The distinction between group and connection does not reflect a biological fact, but represents an abstraction within *iqr* that allows an easier approach to modelling neuronal systems. *iqr* includes a number of standard neuron types such as Linear threshold, Integrate & fire, and Sigmoid neurons (Appendix). The connectionist nature of these neuron models reflect *iqr*'s design philosophy of favouring large-scale neuronal systems and sophisticated connectivity over detailedness and complexity of single elements.

Plug-in Architecture In an *iqr* system, a special role is played by *Modules*, in that they provide a general pur-

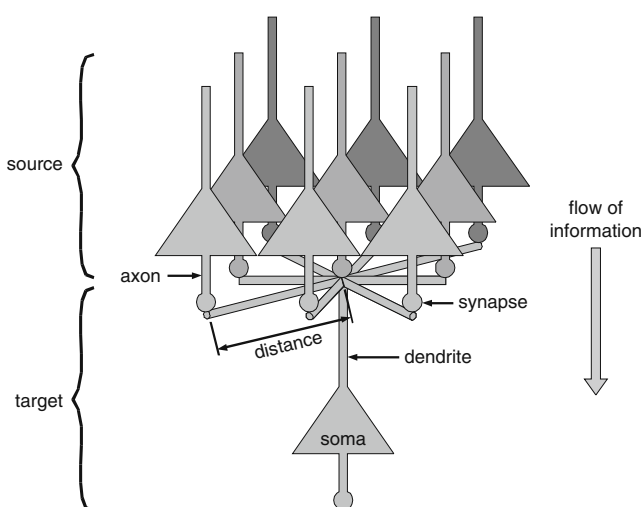


Fig. 3 Illustration of a two dimensional axon-synapse-dendrite nexus where nine pre-synaptic neurons form the receptive field of one post-synaptic cell

pose plug-in architecture to exchange data between the neuronal simulation and an arbitrary external entity. An *iqr Module* is associated with a process (Fig. 2a), and exchanges information by reading values from, and writing values into the groups of that process. All interfaces to hardware are realised as *Modules* (Section “Working with Real-world Bio-robots”). One of the key advantages of the module framework is that it allows to bridge between models defined at different levels of description. This can be used to combine abstract models with neuronally implemented systems, and to bootstrap the development of a neuro-ethological system by initially employing algorithmic components, which are gradually replaced by biologically plausible components.

Modelling of Additional Levels There exist several ways to model additional levels of abstraction in *iqr*. Firstly, subcellular structures can be modelled as groups of “neurons” as was done e.g. for dendritic compartments in Verschure and König (1999). Secondly, an entire simulation application can run inside *iqr* as a module, exchanging information with the main system via the standard module interface. Proof of concept of this approach is that we are running the simulation application *wSim* (Guanella and Verschure 2006) inside *iqr* as a module.

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 the system it is integrated in.

Graphical User Interface for Specifying Neuronal Models

The Main User-interface

To users of any skill level, a graphical interface facilitates the overview over complex models, and yields flexibility in design and exploration of the system. *iqr* allows to do the entire design process of a neuronal model graphically. While the simulation is running, the user can visualise the internal states and change the parameters of system elements.

We will now describe briefly, step-by-step, how to create a model in *iqr*. The procedure starts by creating

a new system using the “File” menu or via the main toolbar (Fig. 4⑦). New groups are created by clicking the group symbol in the diagram toolbar (Fig. 4③), and subsequently clicking in the diagram editing pane (Fig. 4①) at the desired location for the new group. Double clicking the group icon (square), or selecting the “Properties” entry from the right mouse button pop-up menu, brings up the group properties dialogue where the geometry and the neuron type of the group can be specified.

Connections are created by clicking the excitatory, inhibitory, or modulatory connection symbol (Fig. 4③), then clicking on one of the yellow squares at the edge of the source group icon, followed by clicking on one of the yellow squares of the target group icon. As for groups, double clicking the connection and the right pop-up menu gives access to the connection’s properties, allowing to specify the parameters of the connection. To define a connection between groups of different processes, the diagram editing pane (Fig. 4⑤)

is split, and each process is made current in one of the sub panes using the tab-bar (Fig. 4②). Hereafter, the procedure for creating a connection is the same as for a connection between groups of the same process.

Interfaces of the simulation to a robot are defined via the properties of a process. These properties can be accessed either by navigating to the top level diagram pane using the tab-bar, and subsequently double clicking the process icon, or via the right mouse button pop-up menu. The properties dialogue serves to choose the type of interface (*Module*), to set the parameters of the module, and to define which groups of the process receive data from and which send data to the module. After these steps the simulation is ready to be run.

To make the handling of large systems more convenient and to e.g. allow to merge parts from different systems, *iqr* provides full support for cut, copy, paste of processes and groups with or without their associated connections, as well as of connections, within or between instances of *iqr*.

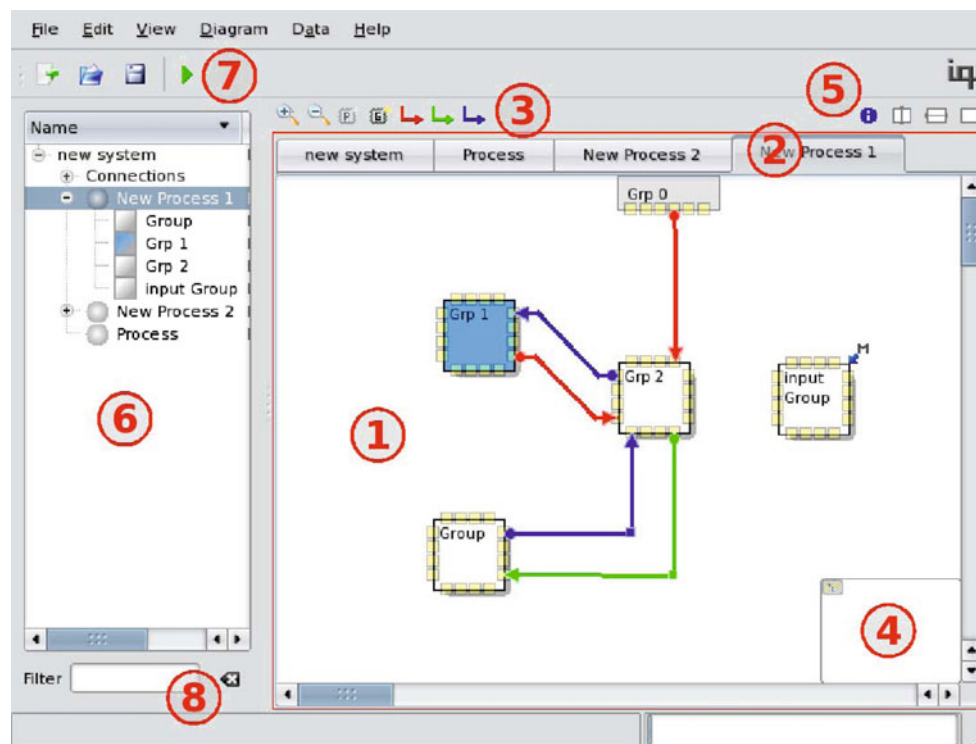
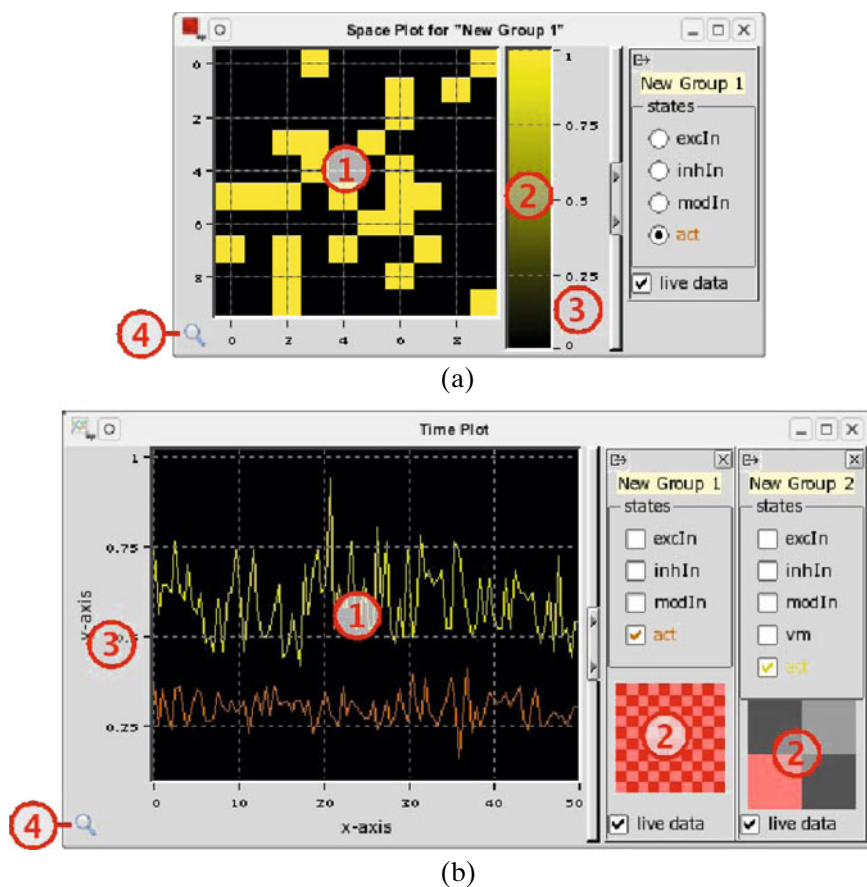


Fig. 4 The *iqr* graphical user-interface. The main display of the system circuit is the diagram editing pane (①). Depending on the level (system or process), a square represents a process or a group. Both, process and group icons, can be assigned a custom colour. Line with arrow heads on the diagram denote connections. A tab-bar (②) is used to switch between diagram panes. The diagram toolbar (③) allows to add processes, groups, and connections to the circuit, and to zoom in and out. The

panner (④) allows to change to the visible section of the pane. The diagram editing pane can be split into two separate views vertically or horizontally (⑤). For direct access to the elements, a browser (⑥) provides a tree like view of the model. A filter can be used (⑧) to search for elements directly in the browser. The main toolbar (⑦) serves to create a new system, open and save system files, and to start the simulation

Fig. 5 **a** *iqr* *Space plot* displays the state of each cell in a group (①). *Space plots* always plot entire groups, i.e. each shows only one group. The value for the selected state is colour coded, and the range indicated in the colour bar (②). **b** A *Time plot* displays the **average** value of the states of an entire group or region of a group against time. A *Time plot* can plot several states, and states from different groups at once. Each state is plotted as a separate trace. A checker board like panel to the right of the plotting area shows which region of a group is plotted and allows to select different states (②)



Visualising and Analysing System Behaviour

iqr provides various options to visualise states of elements of the model. A *Space plot* displays the state of each cell in a group, whereas a *Time plot* shows the average value of one or more states of one or more groups or subregions against time (Fig. 5). To visualise the static and dynamic properties of connections and synapses *Connection plots* are used (Fig. 6). Moreover, this plot type allows for a visual inspection of the con-

nectivity. Plots in *iqr* are running in a thread separate from the main simulation, refreshing at a rate of 20 Hz, independent of the speed of the simulation. If single spikes need to be visualised *iqr* provides the option to synchronise the plots to the simulation.


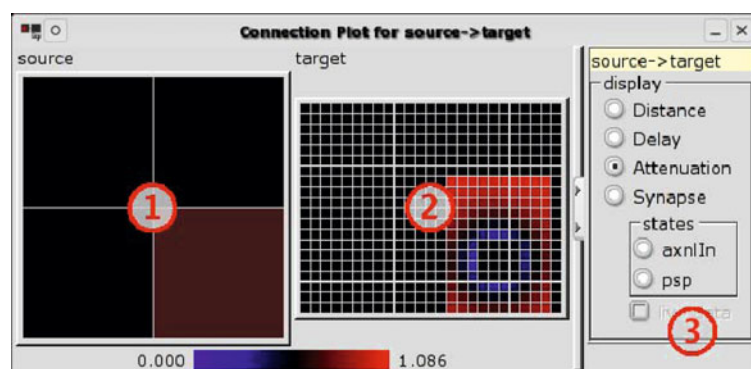
Using the  button it is possible to zoom into *Space plots* and *Time plots* (Fig. 5④). The scaling can be set to “expand only”, “auto scale”, or can be defined manually.

Fig. 6 *iqr* *Connection plot*. The source group (①) is on the left, the target group (②) on the right side. The plot can display static properties of a connection between two groups, i.e. the “Distance” or “Delay”, or the internal states of synapses (③)



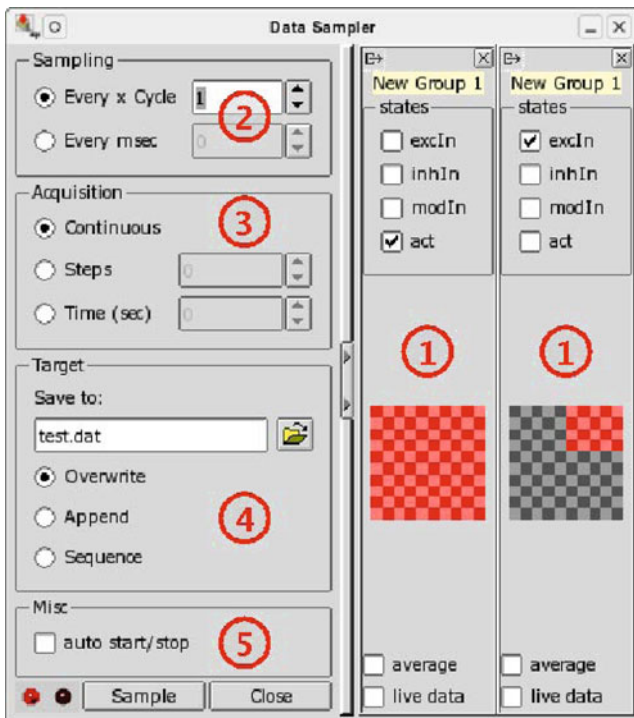


Fig. 7 *Data Sampler*. The panels on the right (①) show which region and which states of a group will be stored, and allow to change between saving the average of the entire group or the value of the individual cells. New groups can be added via drag & drop. The sampling options include setting the sampling rate (②), the acquisition duration (③), where and how to save the data (④). The data acquisition can be set to start automatically with the simulation (⑤)

Recording Data for Off-line Analysis

As mentioned previously, one of the strong points of the synthetic approach is the unlimited access to the internal states of a system. In *iqr* the *Data Sampler* (Fig. 7) serves as the tool with which the user can acquire and record internal states of the model. The *Data Sampler* provides a wide range of options for sampling data ranging from recording the states of single neurons at a high frequency to recording the average state of entire groups of neurons with a low sampling frequency. The frequency of sampling can be set to multiples of the updates of the simulation or defined in milliseconds, whereas the length of the period of data acquisition can be set to continuous, or be limited to a given number of update steps. The format of the file is comma separated plain text (csv). The data can be either written always into the same file at the start of each sampling session, into a new file with an automatically generated file name, based on the specified file name, or can be appended to the end of an existing file. The sampling can be set to start

and stop automatically when the simulation is started and stopped.

The visualising and data collecting functionalities of *iqr* make extensive use of drag & drop. Using the mouse entire groups can be dragged from the *Browser* (Fig. 4⑥), the panel on the right side of *Space plots*, *Time plots*, and *Data Sampler*, or regions of groups from the plotting area of *Space plots*. By dropping this selection onto *Space plots*, *Time plots*, and *Data Sampler* (Section “Recording Data for Off-line Analysis”) can be stored to disk for later retrieval.

Manipulating the System

The *State Manipulation tool* (Fig. 8) serves to change the activity of the neurons in a group by applying a pattern of values. This allows the exploration of the system’s properties by target-orientated manipulations. These patterns are user-defined, and can be stored to and loaded from disk. The “Play Back” options (Fig. 8⑥) specify how often the patterns in the queue

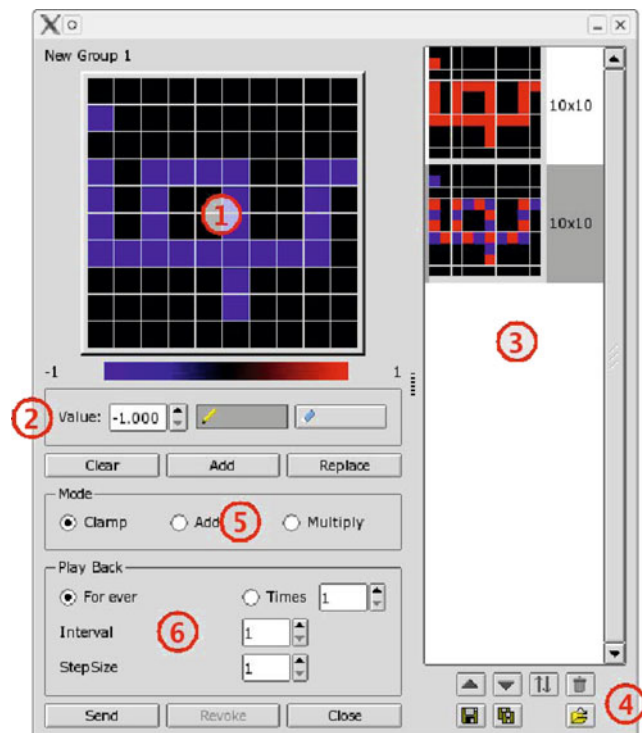



Fig. 8 *State Manipulation tool*. A pattern is created by setting the value in the drawing toolbar (②), selecting the  tool, and drawing the pattern in the drawing area (①). Multiple patterns can be drawn and added to a queue of patterns (③). The order of patterns in the queue can be changed, and the entire queue can be saved to and loaded from a file (④)

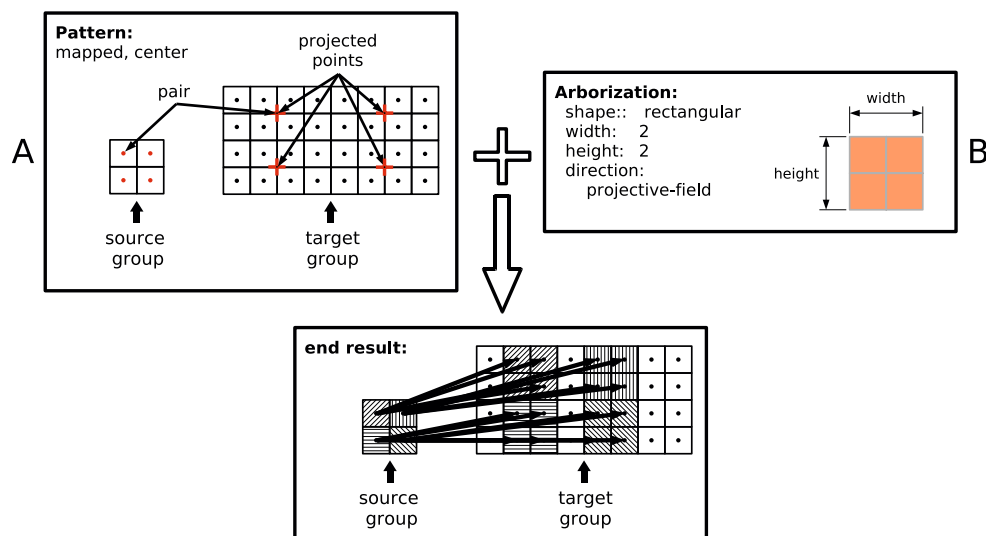


Fig. 9 Definition of the connectivity between two groups. The connectivity *Pattern* (a) is of type “mapped”, meaning that the pairs of points are computed by overlaying and scaling the smaller group (source group in this case) to the larger one. In the example, the *Arborization* (b) is of a rectangular shape with a size

of 2×2 . The **combination**, i.e. the application of the *Arborization* to the projected points in the target group (projective-field), yields the actual connectivity: each of the four neurons in the pre-synaptic group is connected to four neurons in the post-synaptic group, resulting in a total of 16 synapses

will be applied, whereas the “Interval” option determines the delay (i.e. number of time steps to wait) prior to sending the next pattern to the group. The “StepSize” controls which patterns from the queue will be applied (1 means every pattern, 2 means only the first, third, fifth etc.).

The “Send” button applies the patterns to the group, with the effect being either “Clamp”, where the activity of the neurons is set to the values of the pattern, “Add”, where the pattern’s values are added to the activity of the neuron, or “Multiply” where the pattern is multiplied with the activity of the neurons (Fig. 8⁵).

Connectivity

One of the characteristics of the connectivity found in nervous systems is its large variety (Shepherd 2003). A large-scale neuronal systems modelling tool such as *iqr* has to provide the means to define and simulate any arbitrary connectivity scheme in a compact and flexible manner. It must be easy to investigate frequently used schemas, while not barring the specification of non-standard schemas.

Here we will outline the design philosophy behind the implementation of connectivity in *iqr* and will show how connections between groups of neurons are defined, changed and studied.

Specifying the Connectivity

Defining the connectivity between two groups in *iqr* is a two tier process involving two concepts: *Arborization* and *Pattern* (Fig. 9). The *Pattern* (Fig. 9 top left) defines **pairs of points** in the lattice of the pre- and post-synaptic groups. These projected points are 2D coordinates, and can, but do not have to, correspond to the position of neurons. The list of location pairs is referred to as *Pattern*. *iqr* provides several ways to specify the *Pattern* (see below).

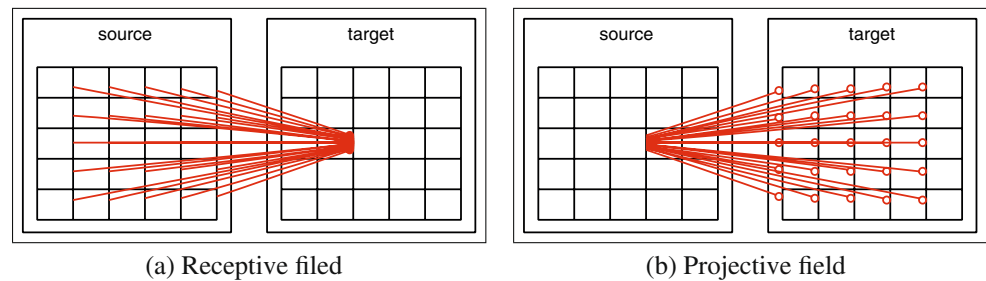
The *Arborization* (Fig. 9 top right) is applied to every projected point defined by the *Pattern*, and hence specifies the neurons that send and receive signals.

iqr allows the user to define the direction of a connection which prescribes whether the *Arborization* is applied to the points defined in the pre- or the post-synaptic group. If applied to the efferent-source group, the effect is a fan-in receptive field (Fig. 10a), while a fan-out, projective field, results when applying the *Arborization* to the afferent-target group (Fig. 10b).

Pattern Types

iqr supports three different kinds of *Patterns*. First a *Pattern* (“For each”) that defines full connectivity, where each cell of the post-synaptic group receives inputs from all the cells of the pre-synaptic one.

Fig. 10 The direction of *Arborizations*. Depending on whether the *Arborization* is applied to the pre- or the post-synaptic group, the resulting structure is a receptive field (RF) (a) or a projective field (PF), respectively (b)



Second the *Pattern* can be a mapping, where the position of the projected points is determined by the position of the cells of the smaller group (fewer neurons), scaled to the size of the larger group (more neurons). Note that a projected point needs not to be at a neuron’s position (Fig. 9 top left).

The aforementioned *Pattern* types are a compact way for specifying a uniform and repetitive connectivity between groups. However, this cannot cover all desired connectivity arrangements. To define individual cell to cell projections, a third *Pattern* type named “Tuples” has been included. With this *Pattern* an arbitrary number of pre-synaptic cells can be associated with an arbitrary number of post-synaptic cells. In this case a tuple t is the combination of m source cells with n target cells: $t = \{(\text{pre}_1, \dots, \text{pre}_m), (\text{post}_1, \dots, \text{post}_n)\}$ and the *Pattern* p itself is the list of tuples: $p = \{t_1, \dots, t_n\}$.

Arborization Types

In Fig. 11 all available *Arborization* types, bar the *Arborization* where all neurons are selected, are shown. Where applicable, the cross indicates the projection point as defined by the *Pattern*.

Delay and Attenuation Functions

From a morphological point of view, a neuron can be divided into axonal and dendritic structures, and

the soma. In i_{qr} , the length of the dendrites is a consequence of the geometry of the layout of the *Arborization* (Fig. 3). The geometry dependent delays and attenuations are therefore dealt with in the dendrites. Axon specific delays and attenuations are implemented at the level of individual synapses. Back-propagating signals reaching the synapse are limited to the dendrites.

i_{qr} uses delay and attenuation functions to compute the delay and the attenuation for each synapse belonging to an *Arborization* (Fig. 12). In most functions, the calculation depends on the size of the *Arborization*, i.e. its height and width.

The delay functions i_{qr} provides are of type linear (1), Gaussian (2), block (3). If the function is random, distance is not taken into account, and the delays are randomly distributed between 0 and max . In the case of a uniform delay, all synapses have the same delay.

$$\text{delay} = ((\text{max} - \text{min}) / \text{dist}_{\text{max}}) * \text{dist} + \text{min} \tag{1}$$

$$\text{delay} = \text{max} - e^{-\frac{\text{dist}^2}{2 * \sigma^2}} * (\text{max} - \text{min}) + \text{min} \tag{2}$$

$$\text{delay} = \begin{cases} \text{min} & \text{dist} \geq \text{width}/2 \\ \text{max} & \text{dist} < \text{width}/2 \end{cases} \tag{3}$$

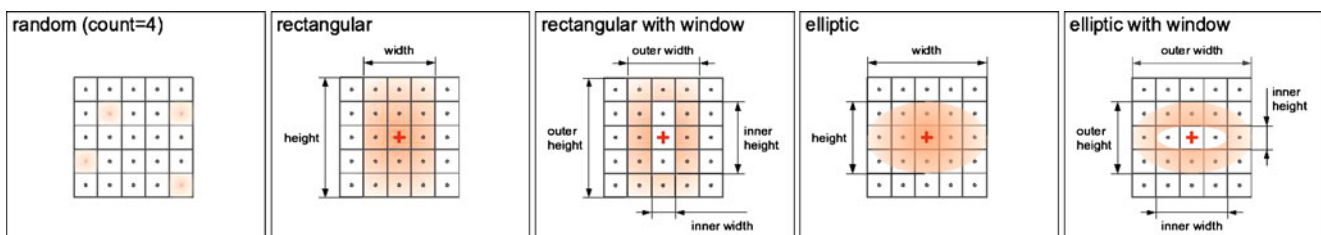


Fig. 11 *Arborization* types available in i_{qr} . Where applicable, the cross indicates the projection point

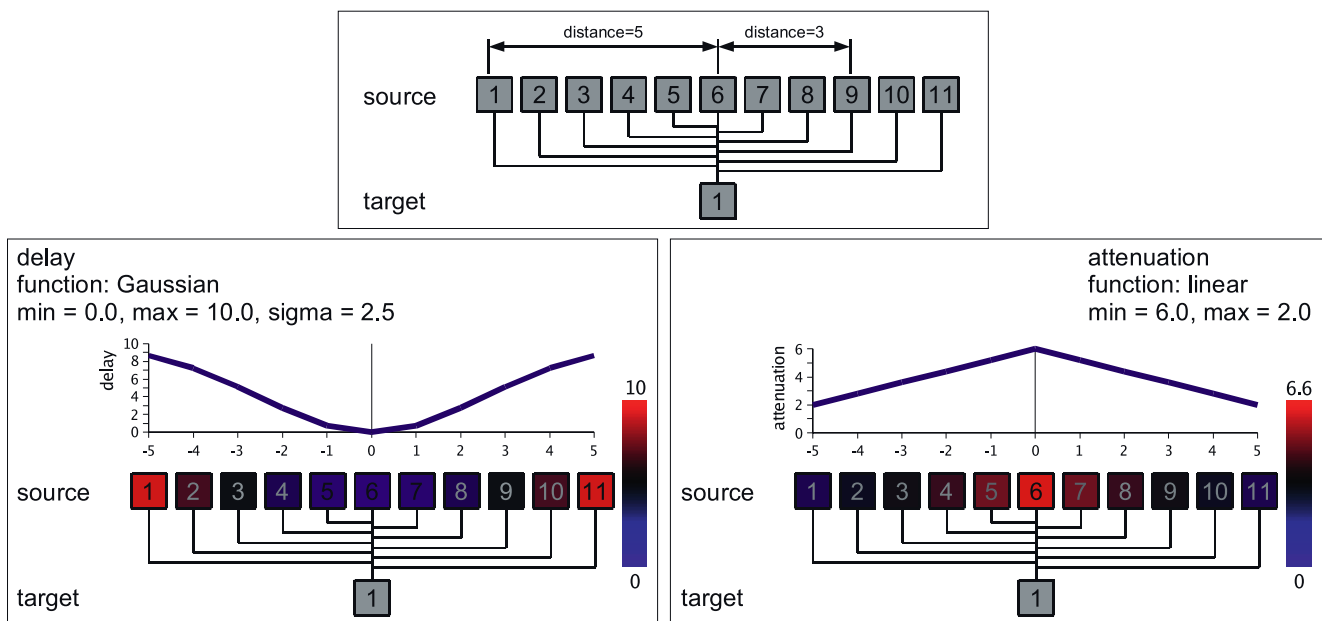


Fig. 12 Example of delay and attenuation functions. (top) The distance is computed as the eccentricity of the sending cell, relative to the position of the receiving cell. The values for delay

(bottom left) and attenuation (bottom right) are based on the distance (as indicated by the x-axis of the function plots)

For the dendritic attenuation (att), the same functions as for the delay are supported. Namely random, uniform, linear (4), Gaussian (5), and block (6).

$$att = att_{max} - ((max - min)/dist_{max}) * dist + min \quad (4)$$

$$att = att_{max} - max - e^{-\frac{dist^2}{2\sigma^2}} * (max - min) + min \quad (5)$$

$$att = att_{max} - \begin{cases} min & dist \geq width/2 \\ max & dist < width/2 \end{cases} \quad (6)$$

Compact Representation

The specification language for connectivity presented here, has major advantages over a more simple specification, e.g. for each synapse individually. On the one hand, the definition is independent of the geometry of the pre- and post synaptic groups (except for *Pattern* “Tuples”), allowing to change the group size without having to redefine the connectivity. On the other hand, the definition is very compact: A system consisting of two groups of 10×10 neurons each (defined to be a torus) linked by a connection with a *Pattern* “For each”, and a rectangular *Arborization* of size 7×6 will result in 490,000 synapses. A representation where each synapse is listed individually results in a system file of

a size in the range of several Megabytes or hundred thousands lines, whereas an *iqr* system file has a size of 4KB or 70 lines.

Working with Real-world Bio-robots

As one of the corner stones of the synthetic approach is the evaluation of neuro-ethological models in the real-world, a tool built for the synthetic approach has to naively support the integration of models with robots. In *iqr* this is accomplished via the *Module* framework, that provides a wide range of predefined interfaces to hardware sensors and actuators. These include modules to control Khepera and e-puck robots (K-Team S.A., Lausanne, Switzerland), Lego MindStorms™, and custom-built blimp-based flying robots. Video images can be fed into the model with modules reading data from PCI frame-grabbers and USB cameras (Fig. 13).

iqr models are interfaced to external devices by specifying mappings between the state of *groups* of neurons and device-specific variables. For input, the value of an external sensor (e.g. a video camera) is mapped onto the state of a group of neurons; for output, the state of a group is used to set the value of a control parameter of the effector (e.g. the speed of a mobile robot). Each process in *iqr* can have one module associated with it (Fig. 2a).

Within the module framework, *iqr* provides a mechanism to automatically generate graphical dia-

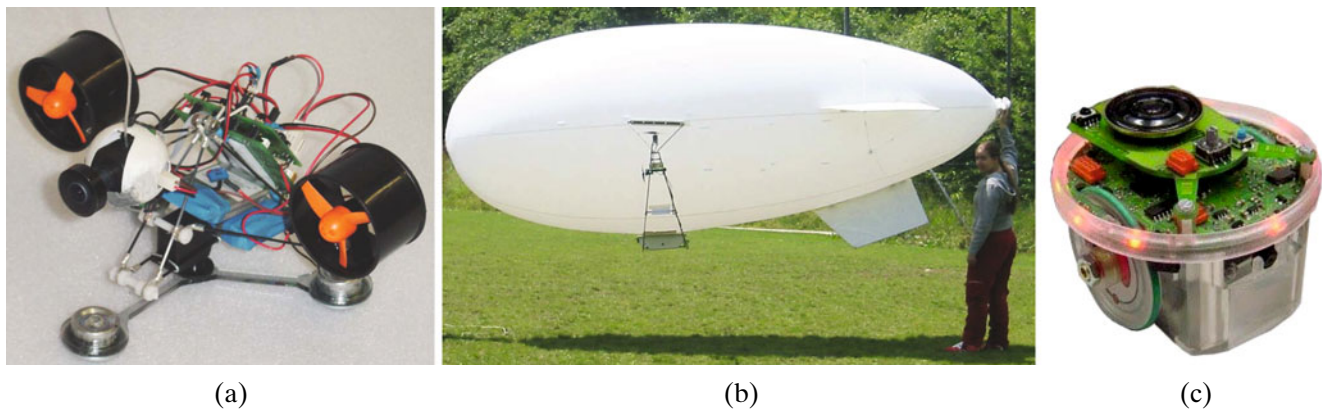


Fig. 13 Selection of robots supported by *iqr*. **a** Passive wheel, impeller driven “Strider” robot. **b** Outdoor blimp (Pyk et al. 2006). **c** e-puck (K-Team S.A., Lausanne, Switzerland)

logues for parameters defined by a module (Fig. 14), which enables the user to change module parameters while the simulation is running without the need to write a specific GUI, or recompiling the module.

Synchronisation Modules can run synchronised, or in their own thread, i.e. their update speed is independent of the update speed of the main simulation. Whereas synchronised modules are important for the replicability of simulation results, asynchronous modules are very useful for the control of robots or the acquisition of data, where a slow hardware device would slow down the entire simulation.

Performance

To be able to control robots in the real-world, simulations have to run at a sufficiently high speed. If a real-world system is relying on video processing, the

simulation has to run faster than the video frame rate of $>25\text{Hz}$ (Fig. 15 red line). At this required update frequency of 25 updates per second, the upper limit of the number of elements in an *iqr* system is $\sim 256,000$ neurons, or $\sim 800,000$ synapses combined with $\sim 5,000$ neurons (Intel^R Pentium^R 4 CPU 3.00GHz, RAM 1GB Fig. 15). If larger systems are implemented, they will need to be run on multiple computers.

To achieve a good performance, *iqr* is implemented in such a way that the storage of the state of individual elements (neurons and synapses) is dissociated from the respective update functions. Concretely, this means that in *iqr*, only one object of a given type is instantiated per group or connection, as opposed to creating e.g. as many neuron objects as the size of a group. This implementation increases performance by avoiding costly virtual table lookups (Stroustrup 1997). The individual states are stored in efficient, vector like structures of type `std::valarray` (Josuttis 1999).

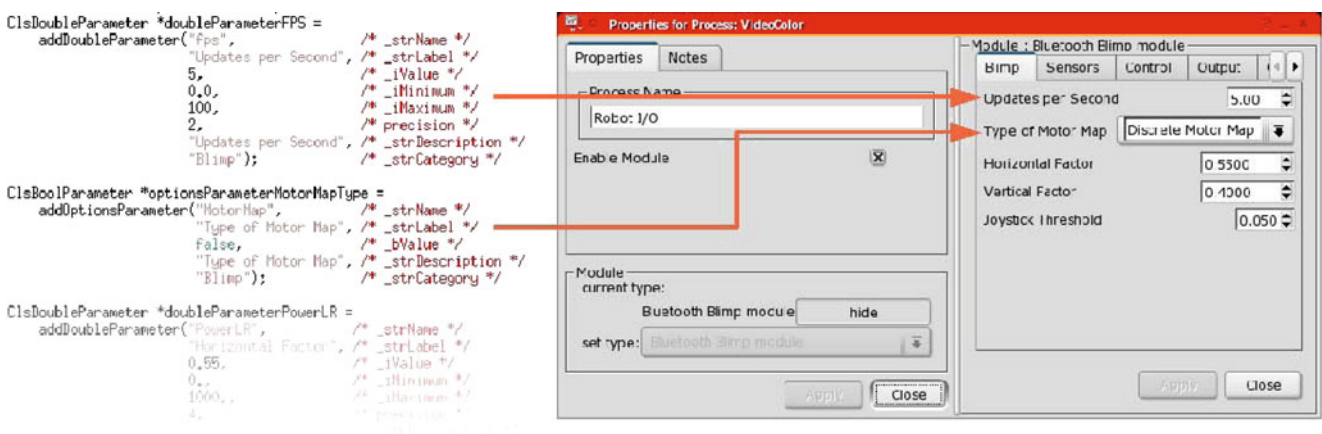


Fig. 14 For parameters defined within the code of a module (left), *iqr* automatically generates a graphical dialogue to access the parameters (right)

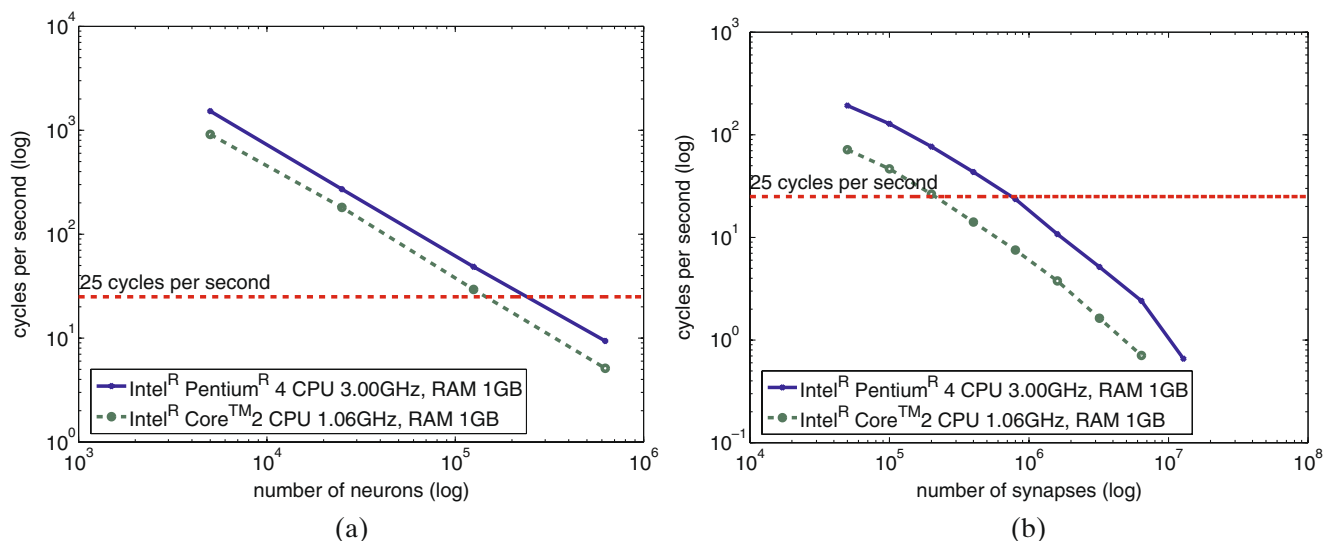


Fig. 15 *iqr* performance. **a** cycles per second vs. number of neurons, **b** cycles per second vs. number of synapses. The neuron type used is “linear threshold”, the synapses were of type “fixed weight”. The red line indicates the 25 cycles per second benchmark

Support for Work-flow of Simulation Experiments

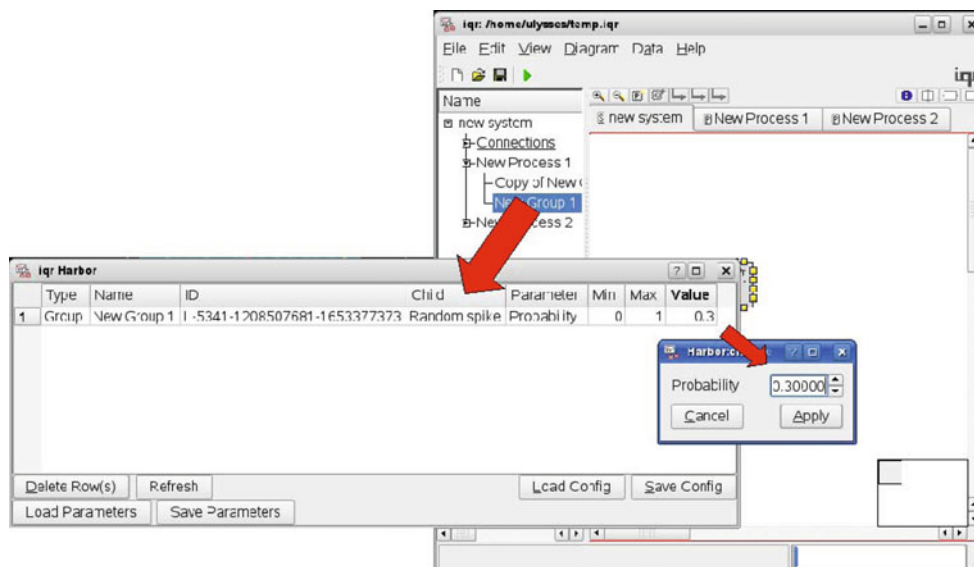
Working with simulations employs a number of generic steps: Designing the system, running the simulation, visualising and analysing the behaviour of the model, perturbing the system, and tuning of parameters. Next to these steps, the automation of experiments and the documentation of the model form important 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 num-

ber of elements. Using the *Harbor*, users can collect system elements such as parameters of neurons and synapses in a central place (Fig. 16), 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*.

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

Fig. 16 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



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*, *start sampler*, *stop sampler*. 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

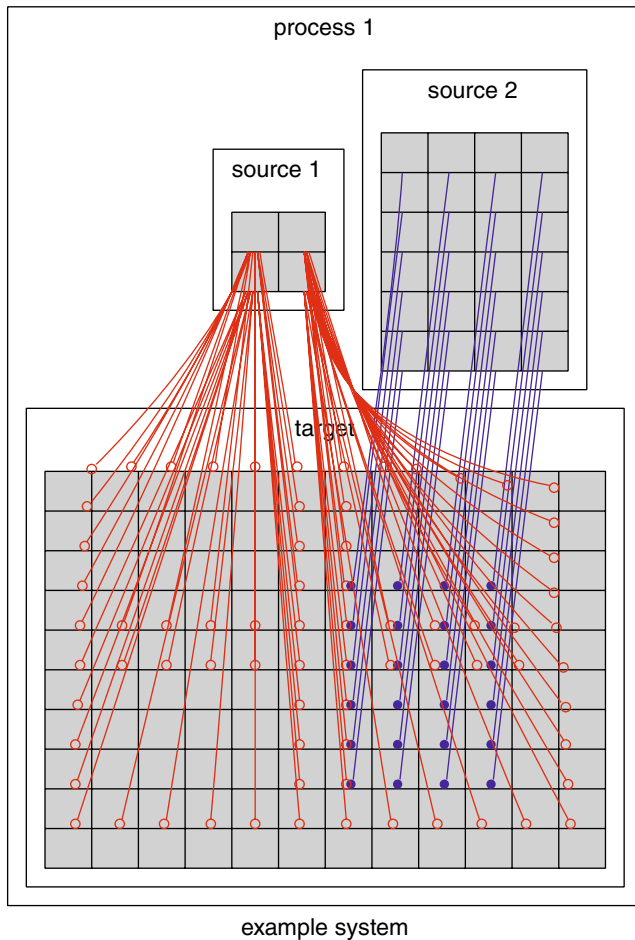


Fig. 17 Example of a transform of an *iqr* system file into a dot language hierarchical directed graph. The system consists of three groups and two connections. Red lines with open circle heads indicate excitatory axon-synapse-dendrite nexuses, blue lines with filled head indicate inhibitory ones (The scripts necessary to do the transform are part of *iqr-tools* package)

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/>) (Fig. 17). Another transform is the creation of system descriptions in the \LaTeX typesetting system.

Description of Projects Using *iqr*

iqr has been used successfully in a number of projects. We will start by describing one on-going project—the construction of a “synthetic insect”—in more detail.

Building a “Synthetic Insect”

The “synthetic insect” project aims at building a large scale, biologically plausible model of an insect, represented by a controlled robot behaving in the real-world. The system synthesises components derived from different insect species, due to the fact that not all required functional components have yet been investigated in a single insect species. The behavioural task of the synthetic insect system is to explore the environment, while searching for the target stimulus, and to return to the point of departure upon either finding the target stimulus or when exceeding a given duration threshold of the exploration phase. The system will be tested using the “Strider” robot (Fig. 22a), an approximation of a flying insect.

As a behavioural model, the synthetic insect system spans the complete nexus of information processing from the input stage (Fig. 18, diamond shaped boxes), to the “cognitive” components (Fig. 18, ellipses), to the generation of behaviours (Fig. 18, rectangular), to the output stage (Fig. 18, parallelogram).

The behavioural elements of the synthetic insect system are organised in a nested hierarchy: The “ex-

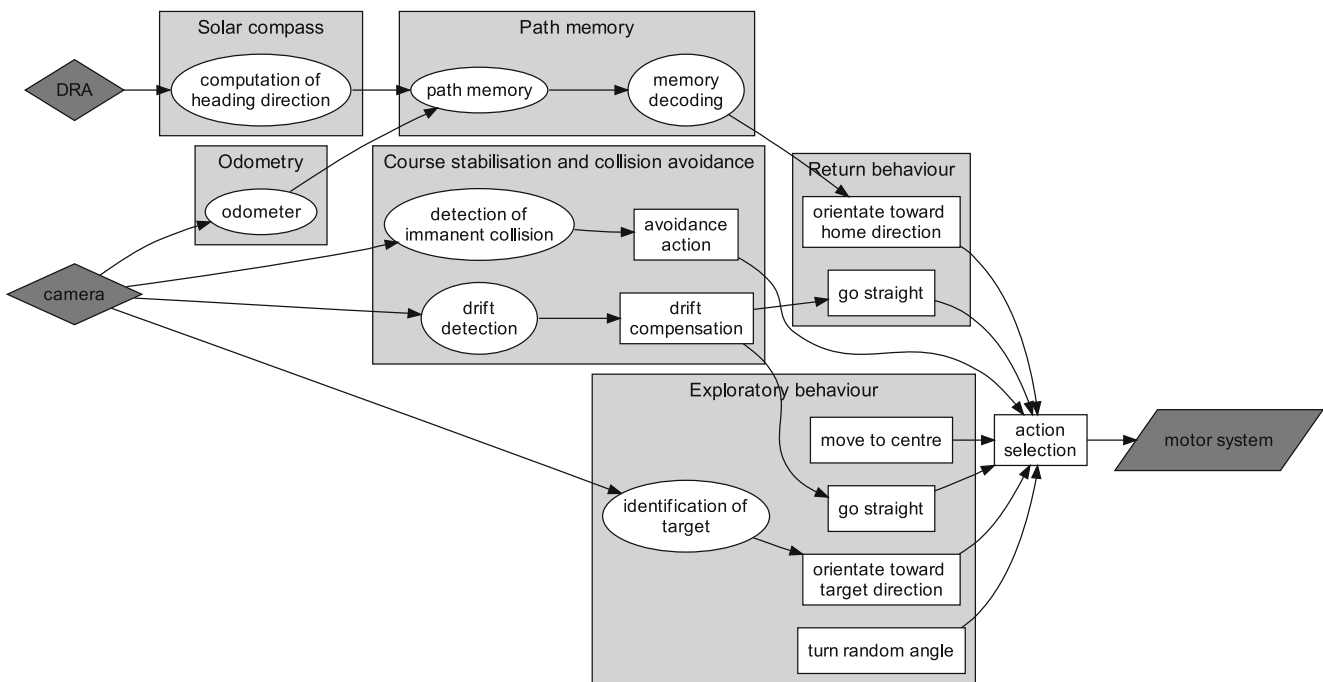


Fig. 18 Components and the flow of information in the “synthetic insect” system. Light grey boxes demarcate subsystems, whereas diamond shapes represent the input stage, ellipses indicate “cognitive” components, and rectangular boxes stand for

behaviours leading to the output stage noted in the parallelogram. DRA = dorsal rim area. (Collision avoidance and course stabilisation subsystems from Bermúdez i Badia et al. (2004) and Bermúdez i Badia et al. (2007b) respectively)

ploratory behaviour” comprises of the aggregated behaviours “random walk”, and “Braitenberg”, which in turn consist of the atomic actions “go straight”, “orientate”, “move to the centre” and “turn random angle”. The “return behaviour” contains the atomic actions “go straight”, and “orientate”. An evasive action is part of the “course stabilisation and collision” behavioural subsystem. As an example, action selection on the one hand covers the effective and efficient management of the atomic actions within one behavioural subsystem, i.e. a prioritisation and resolution of conflicts between the different actions into a coherent overall behaviour. On the other hand, action selection controls the top-down switching between the different complex behaviours. The interaction between behaviours corresponds to reflexes, e.g. ensuring that the collision avoidance reaction has precedence, whereas the top-down control can be understood as the level of “volition”. Thus, these subsystems are mutually interdependent and exchange information. Though they are always active, their output is not always behaviourally relevant.

The synthetic insect system models above outlined nexus of information processing and nested hierarchy of behavioural subsystems and atomic actions. In the course of the development of the synthetic insect we progressed from algorithmic to neurobiolog-

ically grounded implementations. *iqr* facilitates this approach by allowing to implement parts of the model as algorithms in a *Module*.

The “Solar Compass” Insects possess an internal compass system that fulfils a function similar to the head direction cells in mammals. This internal compass system relies on the polarisation pattern of the sky (e-vector orientation) and the luminance distribution. It is therefore referred to as “solar compass”. In our synthetic insect system the solar compass represents the largest single subsystem. In the framework of this subsystem we developed both, hardware realisations of an e-vector and a luminance distribution sensor suitable to fit on the Strider robot (see below), and a neuronal decoding circuit incorporating our present knowledge on the processing of celestial compass information in insects. For the decoding circuit, we developed and employed a number of circuits including the “vector field decoding” motif (see below) and a mechanism for interpolating between groups with different numbers of neurons.

The Path Memory The path memory subsystem where the distance and the direction travelled by the agent are stored, emulates the core of the path integration

system. Here we apply the well established concept of a population code (Georgopoulos et al. 1986) to implement the path memory (Bernardet et al. 2008). In a vector population, each neuron can be understood as vector \vec{v} with norm $\|v\|$ proportional to the firing rate and an angle α given by the neuron's preferred direction. The basic concept of the path memory is the representation of the distance travelled in different directions by a population code. Each column in the “Memory” group represents a vector \vec{v} in a direction j .

Vector Field Decoding Both, the solar compass and the path memory, rely on a circuit that can read out information represented by a population of neurons (“vector field decoding”). Such canonical neuronal circuits, implementing a specific function, can be referred to as motifs (Sporns and Kötter 2004). In *iqr*, the “external process” (Section “Functional Organisation of an *iqr* Model”) mechanism is well suited for developing and deploying such motifs.

Subsequently, we will look at the vector field decoding motif in its application to the reading out of the path memory. The description here is kept very compact, an extended account can be found in Bernardet et al. (2008).

The goal of the memory readout is to calculate the sum vector \vec{s} : $\vec{s} = \sum_{1 \leq i \leq n} \vec{v}_i$. For ease of illustration, we add a group “Memory sum” where each cell receives input from all cells of one column of the “Memory” group (Fig. 20). Hence each neuron in the “Memory sum” has an activity proportional to the number of active neurons in a column of the “Memory” group, and represents a vector \vec{v} with norm $\|v\|$ proportional to

the firing rate and an angle α corresponding to the angle of a column of the “Memory” group. This intermediate step is not a requirement, but helps in constructing the system.

The solution we employ to approximate \vec{s} is a two step process. In a first operation, we project the vectors \vec{v}_j represented by the neurons in the group “Memory sum” onto a set of projection neurons \vec{p}_j (Fig. 19).

Secondly, we apply a MAX operation in this population of projection neurons.

One of the main characteristics of the vector field decoding motif is the connectivity between the groups “Memory sum” and “Vector decode” (Fig. 19). In *iqr*, the required connectivity is achieved by a specific topology of the “Memory sum” group, and one inhibitory and one excitatory connection to the “Vector decode” group. The “Memory sum” has a rectangular topology defined as a vertical cylinder. The inhibitory connection has the following properties: The *Arborization* is defined a rectangular with a window (inner height = 0, inner width = 18, outer height = 1, outer width = 36, direction = RF), whereas the attenuation function is Gaussian with parameters max = -1.3524, min = 1.2177, sigma = 7.1332. For the excitatory connection we use the following parameters: A rectangular *Arborization* (height = 1, width = 18, direction = RF), and an Gaussian attenuation function (max = 2.7848, min = -0.0002, sigma = 9.5347). For both connections we use a *Pattern* of type “mapped”, a uniform delay function, and synapses with a fixed weight of 0.028. The definition of this connectivity is achieved solely through the standard connection definitions in *iqr*, without the need of external tools, and uses 44 lines in the system description file.

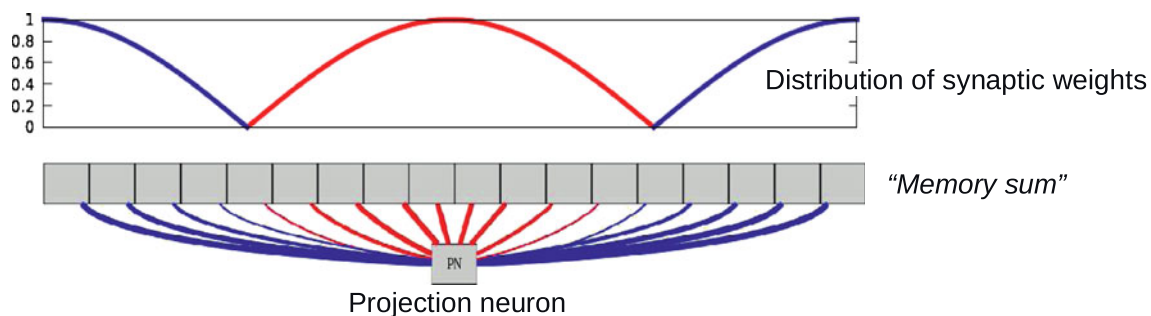


Fig. 19 Illustration of the memory readout mechanism. The figure shows a single projection neuron, which receives input from the “Memory sum” group. Excitatory connections are symbolised by red, inhibitory connections by blue arrows. The synaptic weights are distributed in a Gaussian fashion, approximating a cosine distribution. The top panel shows the weight of the excitatory and inhibitory connections. In the bottom panel the

weight of the connection is represented by the width of the connection. In total the “Vector decode” group (Fig. 20) consists of 36 projection neurons. Since a cosine distribution of synaptic weight is biologically not very plausible, we approximate the cosine distribution with a Gaussian function (figure from Bernardet et al. 2008)

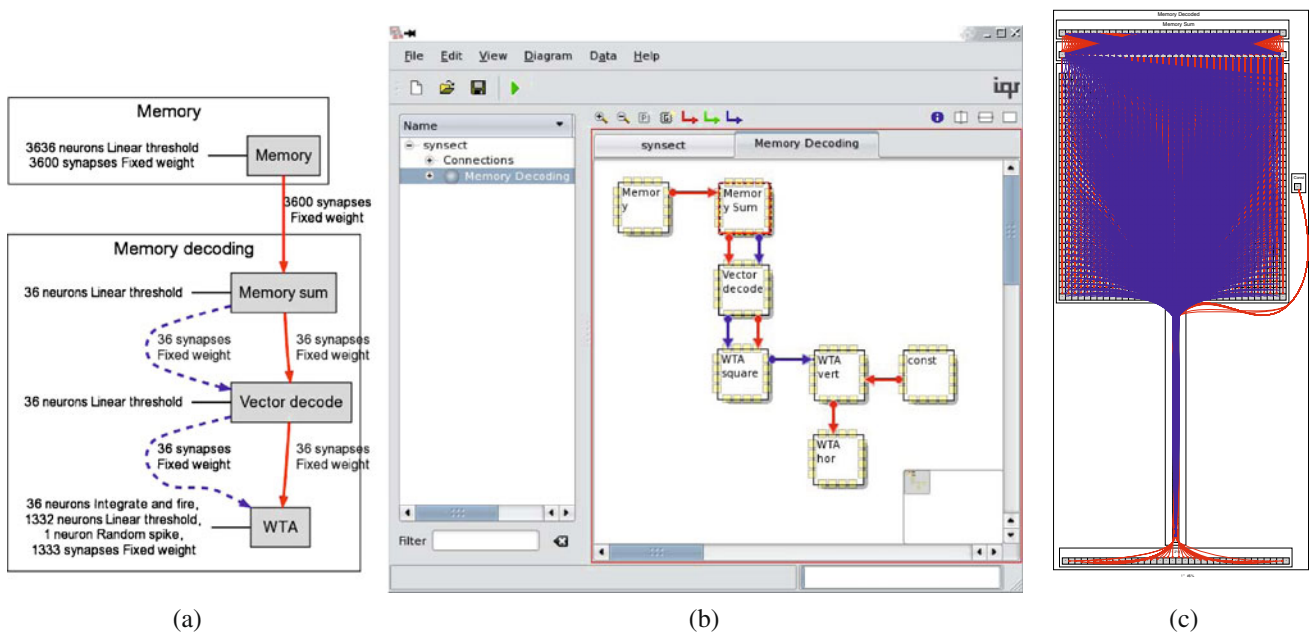


Fig. 20 Three different representations of the architecture of the memory decoding circuit. Excitatory connections are symbolised by *solid red*, inhibitory connections by *(dashed) blue arrows*.

a Abstract view; **b** Graphical user interface in *iqr*; **c** Connectivity at the level of individual synapses (for clarity of representation the “Memory” group is omitted)

Implementation and Evaluation

In total, the synthetic insect system is rather large (Table 1). Inevitably with models of this size, handling and sheer computational power are becoming key issues. To overcome limitations of computational power, we spread the synthetic insect system over two data processors. To this end, the system is split up into four individual *iqr* system files, which in turn are exchanging data over the network using TCP/IP (Fig. 21) and employing *iqr*’s module framework.

Table 1 Statistics of the elements used in the “synthetic insect” system

Element type	Count
Systems	4
Processes	30
Groups	132
Neurons	48,938
Number of groups with neuron of type	
Linear threshold	90
Integrate & fire	12
Linear threshold w. shunting inhibition	14
Sigmoid	7
Random spike	4
Connections	169
Synapses	81,981
Synapse types used in connections	
Fixed weight	169
Modules	31

Experiments are performed in an arena of $6 \times 4.5m$. A camera mounted on the ceiling, in combination with a dedicated computer, is used to track the position of the robot.

The on-line tracking of the robot is fundamental, not only to quantify the performance of the final set-up, but even more so, to closely monitor the operation of the individual components during the development process. For this purpose, an *iqr* module is used that reads the position of the robot over a TCP/IP network connection from a tracking system that in turn runs on the same or a different computer. Thus, the position information of the robot can be saved in the same data file as all the other parameters on the internal states of the model. The communication with the robot is handled by one of the data processors used for the simulation of the system, whereas the video signal from the wireless camera is fed into both computers (Fig. 22b).

Preliminary Results On the behavioural level, the goal of the path integration system is as follows: The robot leaves the start location, explores the environment, thereby searching for a target stimulus, and returns to the point of departure upon either finding the target or when exceeding a duration threshold set for the exploration phase. Figure 23a shows a trajectory of the robot and its different behavioural modes in the experimental arena. The observation of the robot moving

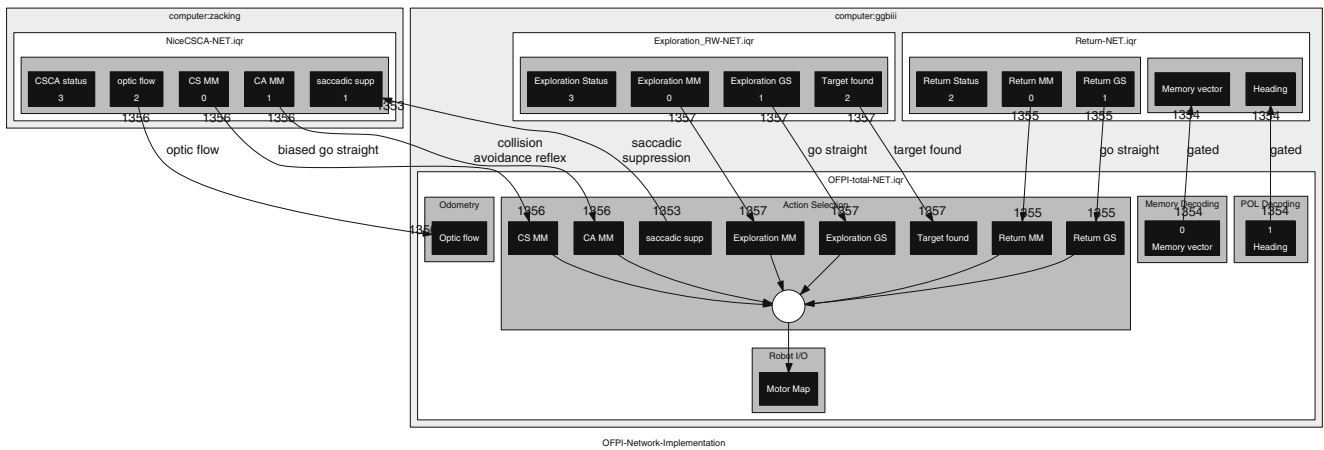


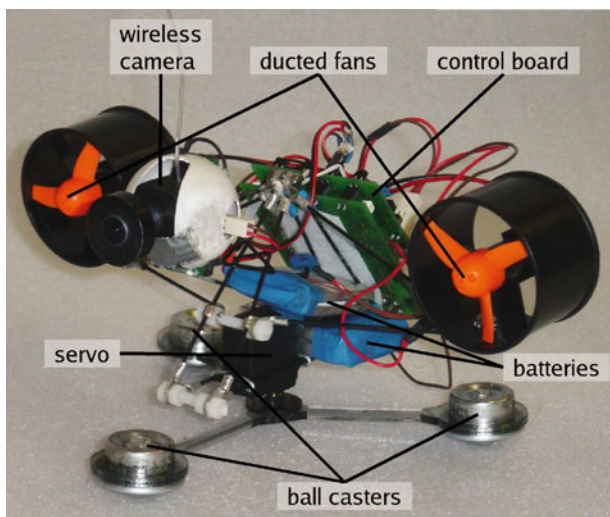
Fig. 21 Distribution of the “synthetic insect” system over four *iqr* instances and two computers, and the exchange of information between the systems. Note that in this view, only the connections

between systems that run on different computers, and some core connections are depicted. *Gray boxes* denote computers, *white rectangles* systems, and *black rectangles* processes

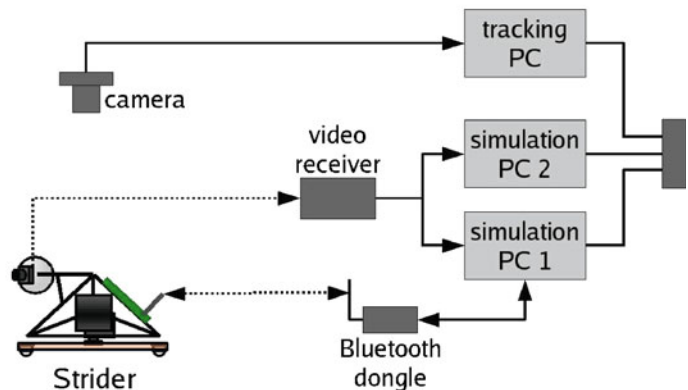
straight when orienting towards the target (Fig. 23a), might seem contradictory at first. The explanation is that orienting corresponds to turning on the spot, which is not captured by the tracking system. The histogram of the angular difference between the memory vector and vector pointing to the start location (Fig. 23b) showed that the direction information stored in the path memory is by and large accurate (mean = 30.94°).

Summary

In this section we described the “synthetic insect” project as an illustration of developing neuronal systems with *iqr*. Comprising 132 groups of neurons, organised into 30 processes (Table 1), this system can be considered large-scale. *iqr* supports this size of systems through its organisation of the model, and



(a)



(b)

Fig. 22 Experimental setup for the “synthetic insect” system. **a** “Strider” robot developed for the “synthetic insect” project. The robot, being equipped with three passive wheels, and propelled by two ducted fans, offers an abstraction of a flying robot in terms of inertia and manoeuvrability. The major advantage over a blimp based robot is the Strider’s small footprint and its ease of deployment. **b** The Strider robot exchanges control

commands and sensors readings via a Bluetooth® link with simulation computer 1. The images from the camera mounted on the Strider are transmitted to a video receiver, and fed into both simulation computers. The tracking computer receives images from a camera mounted to the ceiling, and sends the coordinates of the robot via network to both simulation computers

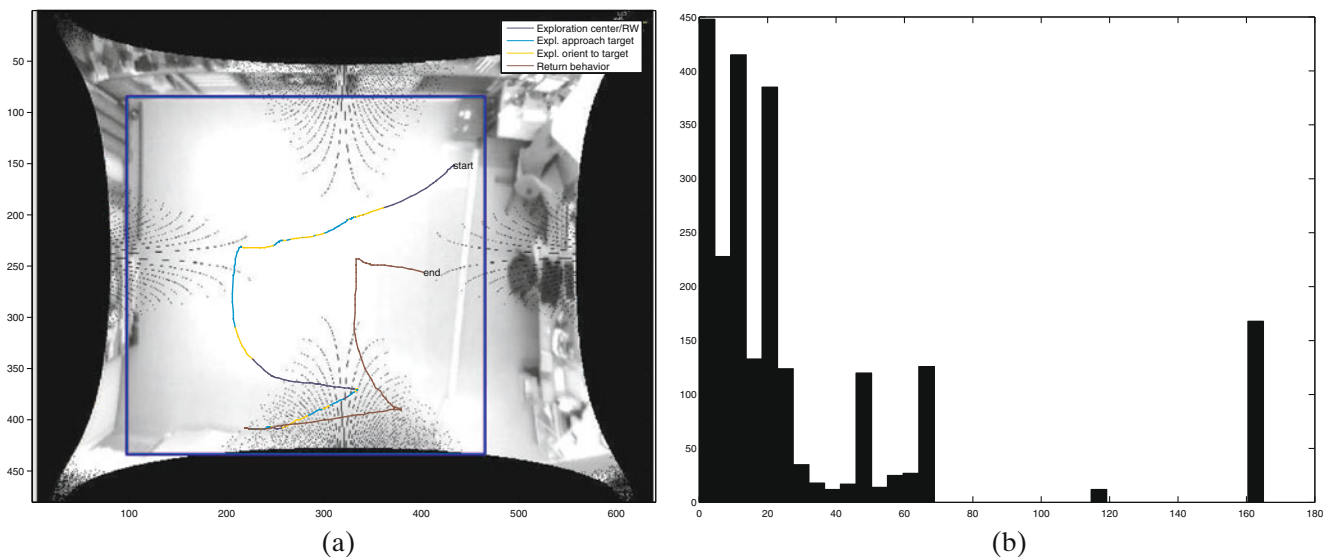


Fig. 23 Example of the behaviour of the strider robot in the experimental arena. **a** Trajectory of the robot exploring the arena, finding the target stimulus (a bright orange patch on the wall), and returning to the start location. The colour of the trajectory

denotes different behavioural modes. **b** Histogram of the angular difference between the vector to the start location stored in the path memory and the actual vector

supports the user by allowing to work with processes that are linked into the system (“external processes”). An example of working at different levels of abstraction is the initial implementation of parts of the model in an algorithmic fashion, which was later substituted by neuronal network implementations. The description of the “vector field decoding” motif should highlight the importance of specific connectivity. The definition of this connectivity was achieved entirely through *iqr*’s native graphical user interface. In accordance with the bio-robotics approach, the system presented here is tested on a real-world robot, controlled in real-time via *iqr*. Regarding the “synthetic insect” system itself, it should be mentioned, that, despite being possibly one of the most inclusive models of insects built so far, the system is not yet complete. We did, for example, not tackle the issue of transferring path integration information from the short-term to the long-term path memory. Yet, only a long-term storage of paths will allow on the one hand to communicate the direction to a food source, as bees do in their waggle dance, and on the other hand to choose from a number of stored paths.

Further Projects

Other project where *iqr* was used include:

The Distributed Adaptive Control (DAC) series of models (Verschure et al. 2003) were built using *iqr*. In these models of learning and memory, a mobile robot formed associations about colour patterns during the

exploration of an arena. The memories stored allowed the robot to reliably navigate to target locations.

iqr is used for models of classical conditioning, including models of learning in the auditory cortex (Sánchez-Montañés et al. 2000) and the cerebellum (Hofstoetter et al. 2002; Herreros et al. 2008)

The project “Ada: the intelligent space” for the Swiss National Exposition (Expo02) was developed to behave as an artificial creature, interacting with visitors in an intuitive manner, and reflecting our present understanding of neurobiological systems (Eng et al. 2003). The Ada project used *iqr* as its basis for the large scale integration of multiple sensors and effectors.

The “eXperience Induction Machine” (XIM) located in Barcelona is one of the most sophisticated mixed-reality environments available today and a further development of the Ada space (Bernardet et al. 2007). A system such as the XIM needs an “operating” system for the integration of sensory information, behaviour regulation, and effector control. *iqr* is used for this for a number of reasons. Firstly, because of the interest in deploying and testing neurobiologically grounded models of perception, cognition and behaviour. Secondly, because the dataflow oriented paradigm of *iqr* fits well with the task of real-time integration of sensory information and effector control. Thirdly, because of the ease of integration of new hardware, and the overall flexibility *iqr* offers.

Intelligent sensor/motor allocation (ISMA) is gaining in importance in many areas of robotics and

autonomous systems. ISMA allows an autonomous entity to allocate its resources for solving the currently most critical task depending on the entity's current state, its sensory input and its acquired knowledge of the world. Using *iqr* (Mathews et al. 2008) implemented and tested an architecture called A-BID. This architecture is guided by a neural network implementation of a selective attention mechanism used to build a probabilistic world model.

The autonomous real-world music composition system RoBoser (Manzolini and Verschure 2005) is an integration of CurvaSom, an algorithmic computer-based composition system, with *iqr* and the DAC model of learning and memory.

iqr was used to develop and deploy models of chemotaxis (Bermúdez i Badia et al. 2007a), collision avoidance (Bermúdez i Badia et al. 2004) and course stabilisation (Bermúdez i Badia et al. 2007b) based on flying insects, in particular the moth. This work is complemented by a model for the neuronal substrate of Path integration memory in arthropods (Bernardet et al. 2008).

In the NEUROChem project (www.neurochem-project.org) *iqr* is used as the main integration platform for different models of olfactory processing in vertebrates and invertebrates. *iqr* was chosen for this purpose because the module framework allows to easily integrate models of different levels of description, developed in various languages.

Conclusion

In this article, we presented *iqr*, a simulation environment for large-scale neuronal systems. With *iqr*, such systems can be developed and simulated graphically without the need to write code. *iqr* was developed to meet the challenges we are faced with when trying to understand neuronal systems, specifically by means of the synthetic approach, i.e. the construction of artificial brain-like systems. The challenge of taking into account multiple levels of abstraction, is met by the *Module* framework which allows to combine models of different levels of abstraction. The system structure of *iqr* models allows to build large-scale models, while the compact and flexible definition of the connectivity between groups of neurons aids in the specification of complex connectivity. Real-world bio-robotics is on the one hand supported by the high performance of the neuronal models, making them suitable for real-time control, and on the other hand through a number of interfaces to robots and sensors that come with *iqr*.

The plug-in architecture of *iqr* allows to easily support custom hardware. To exemplify the use of *iqr*, we elaborate one example, the development of the large-scale, neuro-ethological “synthetic insect” system. *iqr* is still under active development, and the source code of *iqr* is freely accessible under the GNU General Public License. Some of the possible improvements include a scripting interface to automate all aspects of an experiment, new mechanisms to visualise the overall state of a system, and interoperability with other simulators such as NEST (Gewaltig and Diesmann 2007).

Information Sharing Statement

The software presented in this paper is released under the GNU General Public License (GPL). Source and binary packages are available for download from <http://sourceforge.net/projects/iqr/>.

Acknowledgements The authors are grateful to Mark Blanchard, Reto Wyss and Miguel Lechón for their contributions to the development of *iqr*. Important contributions to the neuronal architecture of “synthetic insect” system come from Sergi Bermúdez i Badia. The electronics used in this project was designed and build by Pawel Pyk. The development of *iqr* was supported by the Synthetic Forager (FP7-ICT-217148-SF) project.

Appendix: Predefined Neuron Types in *iqr*

Linear Threshold Neuron Graded potential neurons are modelled using linear threshold cells. The membrane potential of a linear threshold cell i at time $t + 1$, $v_i(t + 1)$, is given by

$$v_i(t + 1) = VmPrs_i v_i(t) + ExcGain_i \sum_{j=1}^m w_{ij} a_j(t - \delta_{ij}) - InhGain_i \sum_{k=1}^n w_{ik} a_k(t - \delta_{ik}) \quad (7)$$

where $VmPrs_i \in \{0, 1\}$ is the persistence of the membrane potential, $ExcGain_i$ and $InhGain_i$ are the gains of the excitatory and inhibitory 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 of the projection from cell j to i and k to i respectively (Table 2).

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

$$a_i(t + 1) = \begin{cases} v_i(t + 1) & \text{with probability } Prob \text{ for } v_i(t + 1) \geq ThSet \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

where $ThSet$ is the membrane potential threshold, and $Prob$ is the probability of activity.

Integrate & Fire Neuron Spiking cells are modelled with an integrate-and-fire cell model. The membrane potential is calculated using Eq. 7. The output activity of an integrate-and-fire cell at time $t + 1$, $a_i(t + 1)$ is given by

$$a_i(t + 1) = \begin{cases} SpikeAmpl & \text{with probability } Prob \text{ for } v_i(t + 1) \geq ThSet \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

where $SpikeAmpl$ is the height of the output spikes, $ThSet$ is the membrane potential threshold, and $Prob$ is the spike probability.

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

$$v'_i(t + 1) = v_i(t + 1) - VmReset \quad (10)$$

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

Sigmoid Neuron The `iqr` sigmoid cell type is based on the perceptron 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 Eq. 7. The output activity, $a_i(t + 1)$ is given by

$$a_i(t + 1) = 0.5 * (1 + \tanh(2 * Slope * (v_i(t + 1) - ThSet))) \quad (11)$$

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

Random Spike Neuron A random spike cell releases a spike per timestep with a user-defined spike probability. The time series of the output spikes forms a Poisson process. 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

$$a_i(t + 1) = \begin{cases} SpikeAmpl & \text{with probability } Prob \\ 0 & \text{otherwise} \end{cases} \quad (12)$$

Table 2 Overview over the parameters of the standard `iqr` neuron types

Random spike	
Probability (<i>Prob</i>)	Probability of a spike occurring during a single time step
Spike amplitude (<i>SpikeAmpl</i>)	Amplitude of each spike
Parameters common to linear threshold, integrate & fire, and sigmoid neurons	
Excitatory gain (<i>ExcGain</i>)	Gain of excitatory inputs. The inputs are summed before being multiplied by this gain.
Inhibitory gain (<i>InhGain</i>)	Gain of inhibitory inputs. The inputs are summed before being multiplied by this gain.
Membrane persistence (<i>VmPrs</i>)	Proportion of the membrane potential remaining after one time step if no input arrives
Clip potential (<i>Clip</i>)	Limits the membrane potential to values between $VmMax$ and $VmMin$
Minimum potential (<i>VmMin</i>)	Minimum value of the membrane potential
Maximum potential (<i>VmMax</i>)	Maximum value of the membrane potential
Linear threshold	
Probability (<i>Prob</i>)	Probability of output occurring during a single time step
Threshold potential (<i>ThSet</i>)	Membrane potential threshold for output activity
Integrate & fire	
Probability (<i>Prob</i>)	Probability of output occurring during a single time step
Threshold potential (<i>ThSet</i>)	Membrane potential threshold for output of a spike
Spike amplitude (<i>SpikeAmpl</i>)	Amplitude of output spikes
Membrane potential reset (<i>VmReset</i>)	Membrane potential reduction after a spike
Sigmoid	
Midpoint	Midpoint of the sigmoid
Slope (<i>Slope</i>)	Slope of the sigmoid

References

- Amari, S., Beltrame, F., Bjaalie, J. G., Dalkara, T., Schutter, E. D., Egan, G. F., et al. (2003). Neuroscience data and tool sharing: A legal and policy framework for neuroinformatics. *Neuroinformatics Journal*, *1*, 149–166.
- Bermúdez i Badia, S., Bernardet, U., Guanella, A., Pyk, P., Knüsel, P., & Verschure, P. (2007a). A biologically based chemo-sensing uav for humanitarian demining. *International Journal of Advanced Robotic Systems*, *4*(2), 187–198.
- Bermúdez i Badia, S., Pyk, P., & Verschure, P. (2007b). A fly-locust based neuronal control system applied to an unmanned aerial vehicle: The invertebrate neuronal principles for course stabilization, altitude control and collision avoidance. *International Journal of Robotics Research*, *26*(7), 759–772. doi:10.1177/0278364907080253.
- Bermúdez i Badia, S., & Verschure, P. F. M. J. (2004). A collision avoidance model based on the lobula giant movement detector neuron of the locust. In J. V. Campenhout (Ed.), *Proceedings of the international joint conference on neural networks (IJCNN'04), Budapest, Hungary* (p. 1757).
- Baernstein, H., & Hull, C. (1931). A mechanical model of the conditioned reflex. *Journal of General Psychology*, *5*, 99–106.
- Bernardet, U., Bermúdez i Badia, S., & Verschure, P. (2007). The experience induction machine and its role in the research on presence. In *The 10th international workshop on presence, 25–27 October*.
- Bernardet, U., Bermúdez i Badia, S., & Verschure, P. (2008). A model for the neuronal substrate of dead reckoning and memory in arthropods: A comparative computational and behavioral study. *Theory in Biosciences*, *127*(2). doi:10.1007/s12064-008-0038-8.
- Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., et al. (2007). Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, *23*(3), 349–398.
- Djurfeldt, M., & Lansner, A. (2007). *Incf workshop report on large-scale modeling*.
- Eng, K., Klein, D., Bähler, A., Bernardet, U., Blanchard, M., Costa, M., et al. (2003). Design for a brain revisited: The neuromorphic design and functionality of the interactive space 'Ada'. *Reviews in the Neurosciences*, *14*, 145–180.
- Georgopoulos, A., Schwartz, A., & Kettner, R. (1986). Neuronal population coding of movement direction. *Science*, *233*, 1416–1419.
- Gewaltig, M. O., & Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, *2*(4), 1430.
- Guanella, A., & Verschure, P. (2006). Artificial neural networks—ICANN 2006. chap. A Model of Grid Cells Based on a Path Integration Mechanism. Springer, Berlin, Heidelberg. doi:10.1007/11840817_77.
- Herreros, I., Zimmerli, L., & Verschure, P. F. M. J. (2008). A biologically based model of the two-phase conditioning: The amygdala, auditory cortex and cerebellum. In: *Proceedings computational and systems neuroscience 2008*.
- Hines, M., & Carnevale, N. (1997). The NEURON simulation environment. *Neural Computation*, *9*, 1179–1209.
- Hofstoetter, C., Mintz, M., & Verschure, P. F. M. J. (2002). The cerebellum in action: A simulation and robotics study. *European Journal of Neuroscience*, *16*, 1361–1376.
- Hull, C. (1952). *A behavior system: An introduction to behavior theory concerning the individual organism*. New Haven: Yale University Press.
- Hull, C., & Baernstein, H. (1929). A mechanical parallel to the conditioned reflex. *Science*, *70*(1801), 14–15.
- Hull, C., & Krueger, R. (1931). An electro-chemical parallel to the conditioned reflex. *Journal of General Psychology*, *5*, 262–269.
- Josuttis, N. M. (1999). *The C++ standard library: A tutorial and reference* (1st ed.). Reading, MA: Addison-Wesley.
- Manzoli, J., & Verschure, P. F. M. J. (2005). Roboser: A real-world composition system. *Computer Music Journal*, *29*(3), 55–74. doi:10.1162/0148926054798133.
- Mathews, Z., Bermúdez i Badia, S., & Verschure, P. (2008). Intelligent motor decision: From selective attention to a bayesian world model. In *Intelligent systems, 2008. IS'08. 4th international IEEE conference* (Vol. 1, pp. 4–8). doi:10.1109/IS.2008.4670418.
- Pyk, P., Bermúdez i Badia, S., Bernardet, U., Knüsel, P., Carlsson, M., Gu, J., et al. (2006). An artificial moth: Chemical source localization using a robot based neuronal model of moth optomotor anemotactic search. *Autonomous Robotics*, *2*(3), 197–213.
- Sánchez-Montañés, M. A., Verschure, P. F. M. J., & König, P. (2000). Local and global gating of synaptic plasticity. *Neural Computation*, *12*, 519–529.
- Shepherd, G. M. (2003). *The synaptic organization of the brain*. Oxford University Press.
- Sporns, O., & Kötter, R. (2004). Motifs in brain networks. *PLoS Biol*, *2*(11), e369. doi:10.1371/journal.pbio.0020369.
- Stroustrup, B. (1997). *The C++ programming language* (3rd ed.). Reading, MA: Addison-Wesley.
- Tolman, E. (1939). Prediction of vicarious trial and error by means of the schematic sowbug. *Psychological Review*, *46*, 318–336.
- Verschure, P. F. M. J. (1997). *Xmorph: A software tool for the synthesis and analysis of neural systems*. Technical report, Institute of Neuroinformatics.
- Verschure, P. F. M. J., & König, P. (1999). On the role of biophysical properties of cortical neurons in binding and segmentation of visual scenes. *Neural Computation*, *11*(5), 1113–1138. doi:10.1162/089976699300016377.
- Verschure, P. F. M. J., Voegtlin, T., & Douglas, R. J. (2003). Environmentally mediated synergy between perception and behaviour in mobile robots. *Nature*, *425*, 620–624.
- Vico, G. (1711). *De antiquissima Italorum sapientia ex linguae originibus eruenda libris tres; On the Most Ancient Wisdom of the Italians Unearthed from the Origins of the Latin Language, including the Disputation with "The Giornale de Letterati D'Italia" [1711]*, translated by L. M. Palmer. (Ithaca: Cornell University Press, 1976). Cornell Paperbacks.