iSpike: A Spiking Neural Interface for the iCub Robot

E Lazdins, A K Fidjeland and D Gamez

Department of Computing, Imperial College London, SW7 2AZ, UK

E-mail: andreas.fidjeland@imperial.ac.uk

Abstract. This paper presents iSpike: a C++ library that interfaces between spiking neural network simulators and the iCub robot. It uses a biologically-inspired approach to convert the robot's sensory information into spikes that are passed to the neural network simulator, and it decodes output spikes from the network into motor signals that are sent to control the robot. Applications of iSpike range from embodied models of the brain to the development of intelligent robots using biologically-inspired spiking neural networks. iSpike is an open source library that is available for free download under the terms of the GPL.

1. Introduction

In recent years there has been increasing interest in spiking neural networks and many people now believe that the precise timing of spikes may be an important part of the neural code [1]. The latest generation of spiking neural simulators now uses CUDA hardware acceleration on commercially available graphics cards to simulate tens of thousands of neurons and millions of connections in real time [2, 3]. Spiking neural networks are being used to construct biologically-inspired models of the brain [4], but with a few exceptions (for example, Krichmar et al. [5]), most of these models have not been embodied, whereas it is now increasingly thought that the brain cannot be fully understood without taking its embodiment and environment into account [6, 7].

Within the field of robotics, a number of sophisticated robots have been developed that are capable of flying, swimming, and terrestrial locomotion, with rich senses that can gather visual, audio, location, tactile, temperature and even air-flow or water-flow information from the surrounding environment. Some work has been done on the use of biologically-inspired spiking neural networks in robotics, but few attempts have exploited hardware-accelerated simulated networks for real-time robot control. One of the key problems has been the lack of an easy way of interfacing between simulator and robot. Such an interface should encode sensory data from the robot into spikes and decode spikes from the simulator into motor commands that are sent to the robot.

To address this gap we have developed the iSpike system, which interfaces between the real or virtual iCub robot and the SpikeStream/NeMo simulators. iSpike encodes

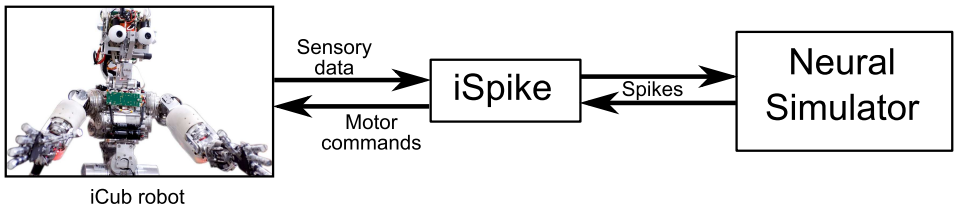


Figure 1. iSpike interfaces between a spiking neural simulator and the iCub robot

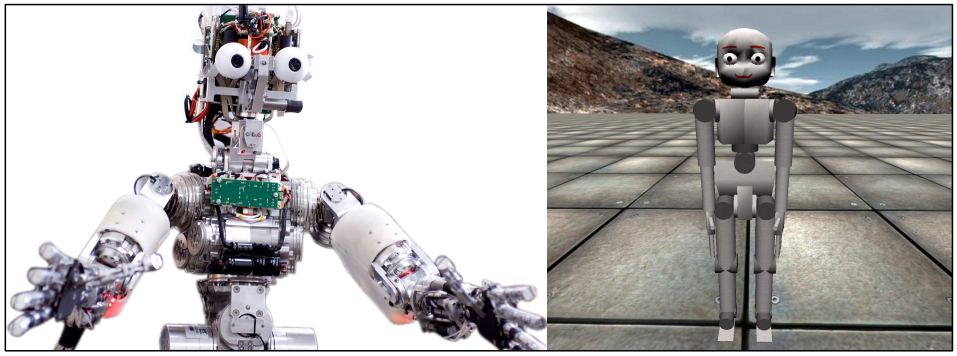


Figure 2. Real and virtual iCub robot

visual and proprioceptive data from the iCub into patterns of spikes that are passed to the simulator, and the spiking output from the simulator is decoded into motor commands that are sent to the robot (Figure 1). iSpike's encoding and decoding has been made as biologically realistic as possible and the system is available for free download under the terms of the GPL (http://ispike.sf.net).

The first part of this paper provides background information about the iCub robot, the tools for neural simulation that we have developed, and previous work on modelling the biological senses and converting data to and from spikes. Section 3 presents an overview of the iSpike architecture, and then Section 4 explains how sensory data from the robot is encoded into spikes, and how spikes from the network are converted back into motor signals that are sent to the robot. An evaluation section contains measurements of iSpike's accuracy and performance, and the paper concludes with a discussion and plans for future work.

2. Background

2.1. The iCub Robot and YARP

The current version of iSpike interfaces with the iCub robot, which was designed to replicate the appearance and capabilities of a two year old child. Its 53 degrees of freedom enable it to carry out complex locomotion and behaviour, and it is equipped with a rich variety of sensors, including digital cameras, gyroscopes, accelerometers, microphones and force/torque sensors [8]. A simulated version of the iCub is also available, which has similar functionality to the real iCub (Figure 2).

iSpike communicates with the real and virtual iCub via YARP (Yet Another Robot Platform). This is a networked robotics middleware platform which provides a communication via a configurable set of ports. The client communicates with particular ports to receive sensory information and sends commands on motor ports to control the robot. This communication with YARP is handled within iSpike: the user just supplies the IP address and port of the YARP nameserver.

2.2. NeMo and SpikeStream Neural Simulators

iSpike works with any neural simulator that can interface with a C++ library. The current implementation targets the NeMo neural simulator and SpikeStream — a graphical interface to NeMo that provides visualization, analysis and network creation tools.

NeMo‡ is a spiking neural simulator that enables high performance real-time simulation of around 100,000 point neurons, delivering up to one billion spikes per second with the use of highly parallel commodity CUDA-enabled graphics processing units (GPUs) [2]. NeMo was originally developed to run the Izhikevich neuron model [9], and it has been recently extended to support arbitrary neuron models as well as Kuramoto oscillators [10]. Learning is supported through the spike time dependant plasticity (STDP) rule put forward by Song et al. [11]. NeMo is an open source project that is distributed as a C++ class library with APIs for Python, PyNN, Matlab, and pure C, and it can interface with iSpike within a C++ program that constructs the network in NeMo, initializes iSpike with appropriate input and output channels, and passes spikes to and from iSpike at each time step.

SpikeStream§ provides a sophisticated 3D graphical interface that enables the user to create and edit neuron and connection groups and control simulation and archiving (Figure 3). Key functions, such as network creation, simulation and analysis, are implemented as plugins, which makes it easy to customise and extend the functionality. Simulations are carried out using the hardware acceleration provided by NeMo, and SpikeStream includes analysis plugins for measuring state-based ϕ and liveliness in the network [12]. A MySQL database is used as the storage back-end, and SpikeStream is written in C++ using Qt for the GUI.

To facilitate iSpike integration, SpikeStream includes a wrapper plugin that injects output spikes from iSpike into the network and passes spikes from the network into iSpike (Figure 3). This wrapper enables the user to connect different iSpike channels to different parts of the network and to configure the properties of the channels.

2.3. Previous Work

The most similar system to iSpike is the interface that was created for the CRONOS and SIMNOS robots [13], which encoded visual and proprioceptive data from the robots into

‡ NeMo is available for download at: http://nemosim.sf.net.
§ SpikeStream is available for download at: http://spikestream.sf.net.

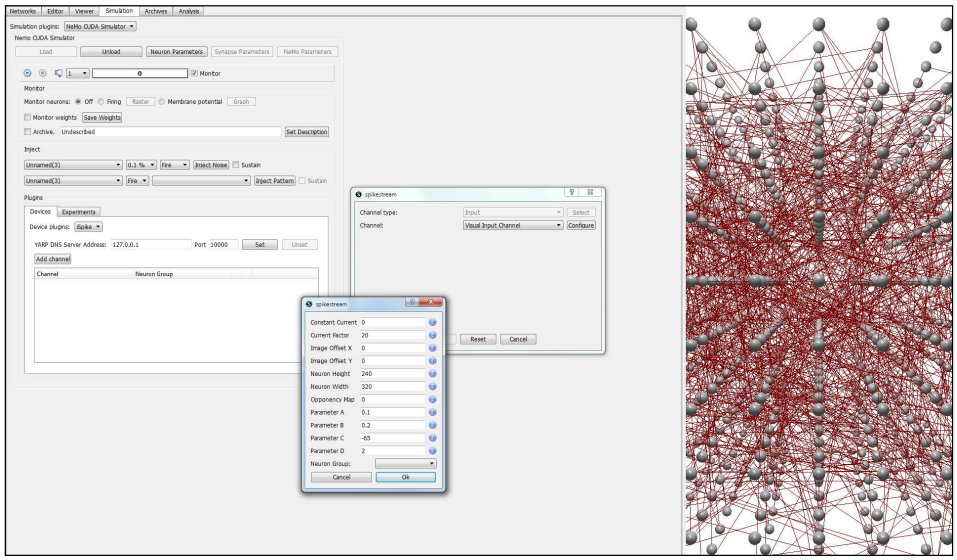


Figure 3. SpikeStream GUI showing wrapper for iSpike

spikes that were passed to a spiking neural network simulated in SpikeStream. Spiking motor output from the network was transformed back into real values that were used to control the robots. This system was used to develop a spiking neural network that controlled the eye movements of SIMNOS, learnt associations between motor output and visual input, and used models of imagination and emotion to avoid negative stimuli [14]. A second related project is the spiking interface developed by Bouganis and Shanahan [15], which converted the iCub's proprioceptive signals into spikes, and converted spikes from a network back into motor commands that were sent to the iCub. Bouganis and Shanahan used this interface to develop a spiking network that autonomously learnt to control a robot arm after an initial period of motor babbling.

A number of different approaches have been proposed for converting data to and from spikes [16, 17], and different coding and decoding strategies have been investigated by Novellino et al. [18] using a bidirectional neural interface between in vitro rat neurons and a two wheeled robot. There has also been a large amount of related work on retina modelling using software — for example, [19] — and hardware, for example [20, 21]. A key limitation of much of this work is that the software for spike encoding and decoding has not been made generally available or that it has been based around proprietary hardware or robotic systems that are not widely used.

3. System Overview

iSpike is implemented as a cross-platform C++ library. The current version supports the encoding of proprioceptive data from joint angles into spikes, the decoding of spikes into joint angles and the conversion of camera data into spikes in a retina-inspired manner. The architecture of iSpike is illustrated in Figure 4.

Internally iSpike consists of the following components:

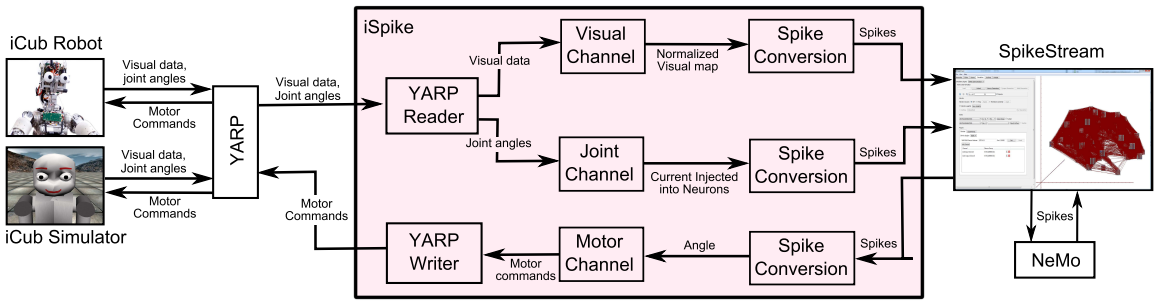


Figure 4. iSpike architecture and its interface with YARP/iCub and SpikeStream/NeMo

- *Reader*. Extracts sensory data of a given type from a given location. The current release of iSpike has readers that can extract file data as well as readers that can access visual data and joint angles from YARP. Readers run as separate threads that buffer the latest data.
- *Writer*. Outputs data of a given type to a given destination. The current release has writers that output data to file as well as writers that send motor commands to the iCub using YARP.
- *Input Channel*. Receives sensory data from a Reader of a given type and transforms it into a spike representation, which is passed to the neural simulator. For example, a JointInputChannel converts the angular data prepared by an AngleReader into spikes. The spike conversion process depends on simulated Izhikevich neurons, which are stepped in synchrony with the neural network simulator (see Section 4).
- *Output Channel*. Receives a spike pattern from the neural simulator, converts it into an appropriate format, and uses a Writer to deliver it to a predefined destination. So, for example, a JointOutputChannel converts neuron spike patterns into joint angle motor commands and uses an AngleWriter to deliver these to the iCub. The spike conversion process depends on simulated Izhikevich neurons, which are stepped in synchrony with the neural network simulator (see Section 4).

The next section describes how visual and proprioceptive data from the iCub is pre-processed and converted into spikes, and how spikes from the network are converted into motor commands that are sent to the robot.

4. Sensory and Motor Spike Encoding and Decoding

This section describes how iSpike pre-processes data from the robot and converts it to and from spikes. To achieve a more biologically realistic conversion it is necessary to understand the encoding and decoding methods that are used by the body to transform analogue sensory information into a spike pattern, and to transform a spike pattern into a movement of the body. This is an active research field and there is much debate about

the methods that are used by the sensory and motor systems of different biological organisms. The most popular interpretations of neural coding are [1]:

- *Rate coding.* In this approach, the rate at which a neuron fires is proportional or inversely proportional to the intensity of the stimulus. The rate is interpreted as either the average firing rate over time, the average firing rate over several repetitions of an experiment, or the average rate over a population of neurons. There is evidence that rate-based coding is used in the stretch receptor of a muscle spindle [22] as well as in the auditory cortex of marmoset monkeys [23]. A common criticism of rate-based encoding schemes is that either a considerable amount of time or a large number of neurons is required to obtain an accurate value for the mean firing rate [24].
- *Temporal coding.* In this encoding scheme the intensity of the stimulus is related to the absolute latency of the first spike to arrive, or to the relative latency between the first spikes arriving at a particular neuron. This approach was used by Gollisch et al. [25] to accurately reconstruct the image displayed to a salamander retina. A major advantage of this method is that a small number of spikes is sufficient to accurately derive the intensity of the original stimulus, which could account for the brain's low latency reaction times to environmental stimuli. A disadvantage is that the decoded value can be significantly affected by jitter.
- *Rank order coding.* This uses the order of arrival of spikes to encode the data, with each ordering encoding a particular value related to the original stimulus, such as its intensity level. Using this method 6 neurons can encode $6! = 720$ distinct values. This approach is immune to noisy temporal jitter, it is invariant to changes in contrast and luminance (when encoding images), and it can be used to encode a large amount of information very rapidly [24]. While there is no clear evidence of rank order coding in biological systems, it has been successfully used in modelling work to encode and decode images [26].
- *Population coding.* In this method the sensory signal is encoded as the collective activity of a population of neurons (the population vector). There is evidence that population coding is used for some types of motor control in the brain — for example, Georgopoulos et al. [27] showed that the direction of movement of a monkey's arm was strongly correlated with the population vector value received from neuron groups in the primary visual cortex. While population coding is typically associated with a rate-based code, it can also be applied to other forms of coding — for example, the average time to spike of a whole population could be used instead of the time to spike of individual neurons within the population.

The remaining parts of this section describe the spike encoding and decoding methods used by iSpike, which attempt to be compatible with as many of these interpretations of the neural code as possible.

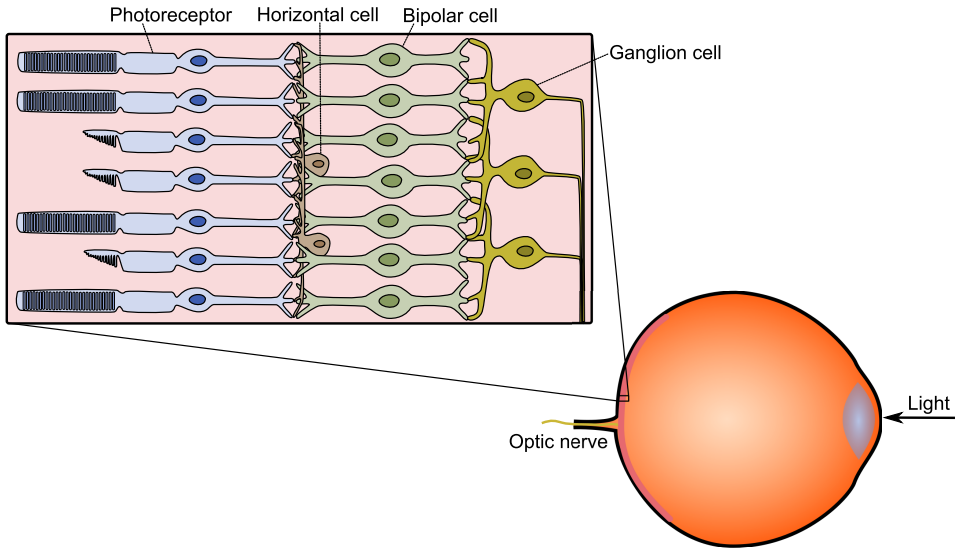


Figure 5. Simplified illustration of the human retina showing some of the information-processing cells

4.1. Biological Vision

Visual information enters the human eye in the form of electromagnetic waves with a variety of wavelengths and amplitudes. These waves are focused by the lens onto the retina at the back of the eye, where they travel through a number of layers until they reach the photoreceptor cells, which are the starting point for retinal visual processing (Figure 5.

Photoreceptors come in two types called rods and cones. Rods have a similar response to all wavelengths of visible light; cone photoreceptors come in three varieties that are tuned to light with a wavelength of 430 nm (blue), 530 nm (green) or 560 nm (red). The central or fovea region of the retina is almost exclusively covered in cones, whereas the periphery is mostly covered with rods. Rods are over 1000 times more sensitive to light than cones, and during night time it is mostly these that contribute to vision. Conversely, during daytime lighting conditions, the cones are mainly used, as the light intensity is too high for the rod photoreceptors to operate. The current implementation of iSpike ignores the presence of rod photoreceptors, as daylight conditions are assumed throughout.

When light hits a rod or cone photoreceptor, neurotransmitter is released and the signal is transmitted through bipolar and horizontal cells to ganglion cells. The wiring and response characteristics of bipolar and horizontal cells causes many ganglion cells to be excited by a particular colour of light at the centre of their receptive field and inhibited by a different colour of light at the periphery of their receptive field. So, for example, a Red+Green- (R+G-) ganglion cell is excited by a point of red light at the centre of its receptive field and inhibited by an annulus of green light at the periphery of its receptive field. The response characteristics of the most common types of colour-opponent ganglion cells are shown in Figure 6.

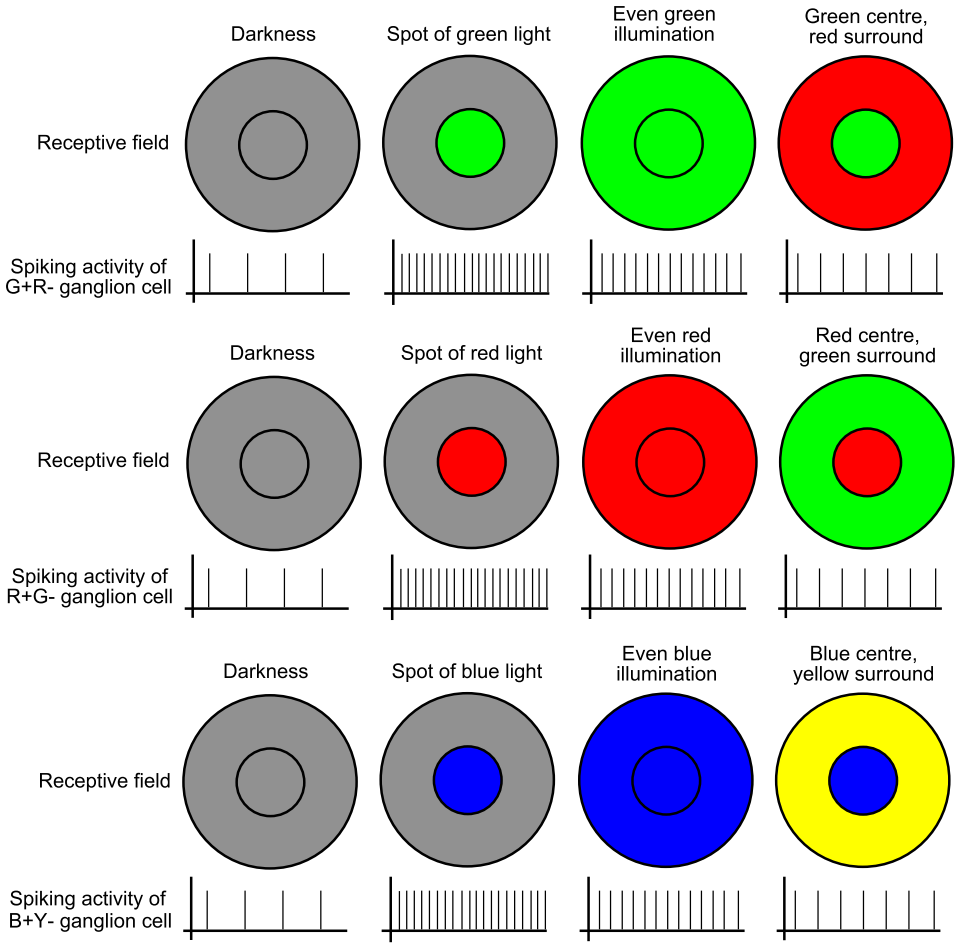


Figure 6. Receptive fields and response characteristics of different types of colour-opponent retinal ganglion cells. Adapted from [28].

Ganglion cells are also classified according to the temporal characteristics of their response. Approximately 5% are magnocellular cells, which respond to stimulation with a transient burst of action potentials and are particularly sensitive to movement. 90% are parvocellular cells, which have smaller receptive fields and produce a steady stream of action potentials for as long as the stimulus is present. Only parvocellular cells exhibit the colour opponency shown in Figure 6, and it is these that are modelled in the current version of iSpike.‖

4.2. iSpike Encoding of Visual Data into Spikes

The encoding of visual data into spikes is carried out in two stages. The image is pre-processed in a manner mimicking the transformations that take place in the biological retina. This results in a set of maps that are optionally normalized between 0 and 1, and then converted into spikes by feeding the analogue values as currents into a layer of Izhikevich neurons [9].

‖ This high level summary of the human visual system is based on [28]

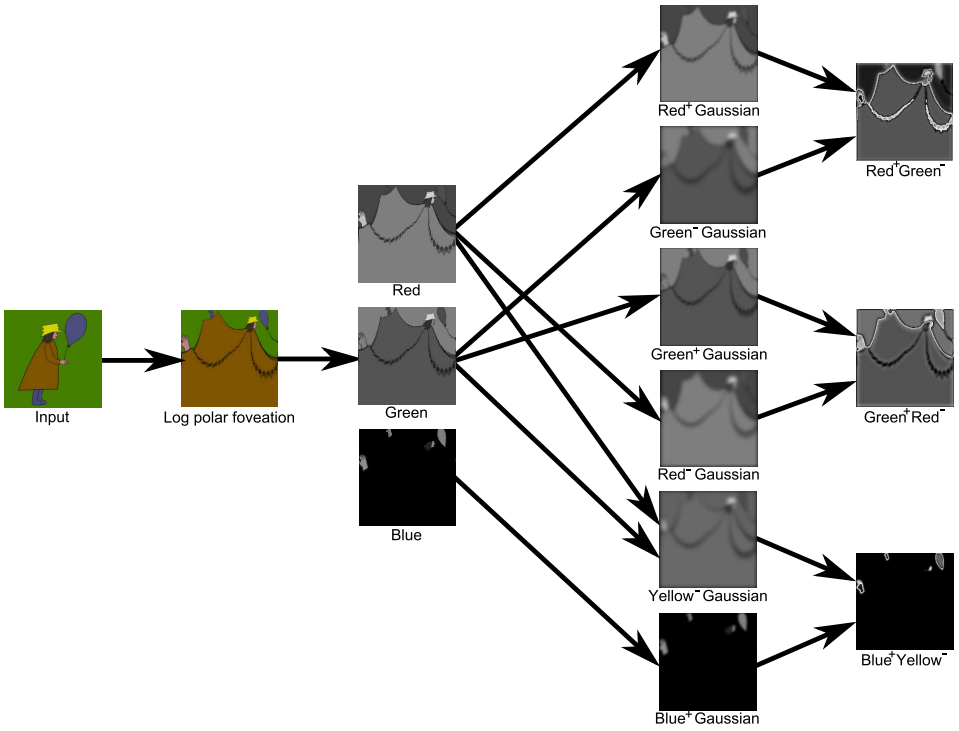


Figure 7. Visual pre-processing. In the first pre-processing stage each frame in the stream of 320×240 images from the iCub's cameras is re-sampled into log polar form to mimic the foveation of the retina. This foveated representation is transformed into colour-opponent visual maps using a difference-of-Gaussians method.

The first stage of the image pre-processing is the foveation of the rectangular image received from the iCub. A number of ways of modelling the foveation of the retina have been put forward, including $log(z)$, $log(z + a)$ and Wilson's model [29]. While Wilson's model is the most biologically realistic, it is also potentially the most computationally expensive, and the current implementation of iSpike uses the simpler log$z$ model, which just involves a log polar resampling of the original image. The colour-opponent ganglion cells are simulated by the difference-of-Gaussians method created by Enroth–Cugell and Robson [30]. In the case of Red+Green- cells, red and green versions of the original image are created, and then a Gaussian filter is applied using different standard deviations for each image. The final R+G- image is obtained by subtracting the Gaussian green from the Gaussian red image, as illustrated in Figure 7.

After an optional normalization to the range 0-1, the final stage of visual processing is the conversion of the foveated colour-opponent maps into spikes. In the retina the early stages of visual processing in bipolar and horizontal cells are analogue, with the final spiking output being generated by the ganglion cells. This is modelled by feeding the analogue visual maps into a 2–dimensional array of Izhikevich neurons [9], with the current entering each neuron being proportional to the intensity of the corresponding pixel (Figure 8). The resulting neuron spiking activity is passed to the neural simulator.

The advantage of this approach is that it encodes analogue values into spikes in a

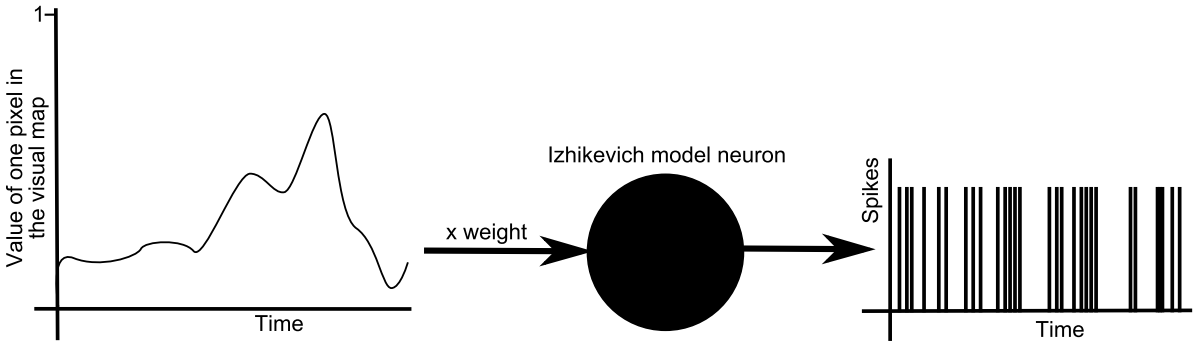


Figure 8. Conversion of visual data into spikes. After the visual maps have been optionally normalized, each intensity value is converted into a sequence of spikes by feeding it as input current into an Izhikevich neuron. These neurons can be configured by the user to produce a wide range of spiking responses — for example, regular spiking, chattering, and intrinsically bursting — to match the interface requirements of the cognitive architecture. The resulting spikes are added to a buffer where they can be asynchronously read by the spiking neural simulator.

way that is compatible with temporal, rank order and rate-based coding. The neuron that receives the most current will be the first to spike and it will also fire at the highest rate. The neuron with the most current will also be likely to fire before the one with the second highest current and so on, thus producing a rank order code. Configuration of neuron parameters can be used to control what type of spike code to employ.

4.3. Biological Proprioception

Proprioception is the somatic sensory system responsible for the identification of body part ownership and for providing information about the position and movement of individual body parts. Most of the muscles in the human body contain specialised structures called muscle spindles that send information about changes in the muscle's length, which is directly related to the joint position. Muscle tension is detected by the Golgi tendon organ, which acts like a strain gauge.

4.4. iSpike Model of Biological Proprioception

The iCub robot provides information about the angles of each of its joints. While these could be converted into spikes using a rate or latency code in a single neuron, this would take too long to read accurately or result in significant error. Instead, a population-based approach is used, in which a group of neurons spans the range of possible angles. When a joint angle is received from the robot it is converted into a vector of input current values by convolving it with a Gaussian function. These current values are multiplied by a parameterized weight and fed into the neurons, producing a spike pattern that is delivered to the neural simulator (Figure 9). This use of currents pumped into Izhikevich neurons is compatible with several interpretations of the neural code, since either the rate, temporal difference or rank order in which the neurons fire

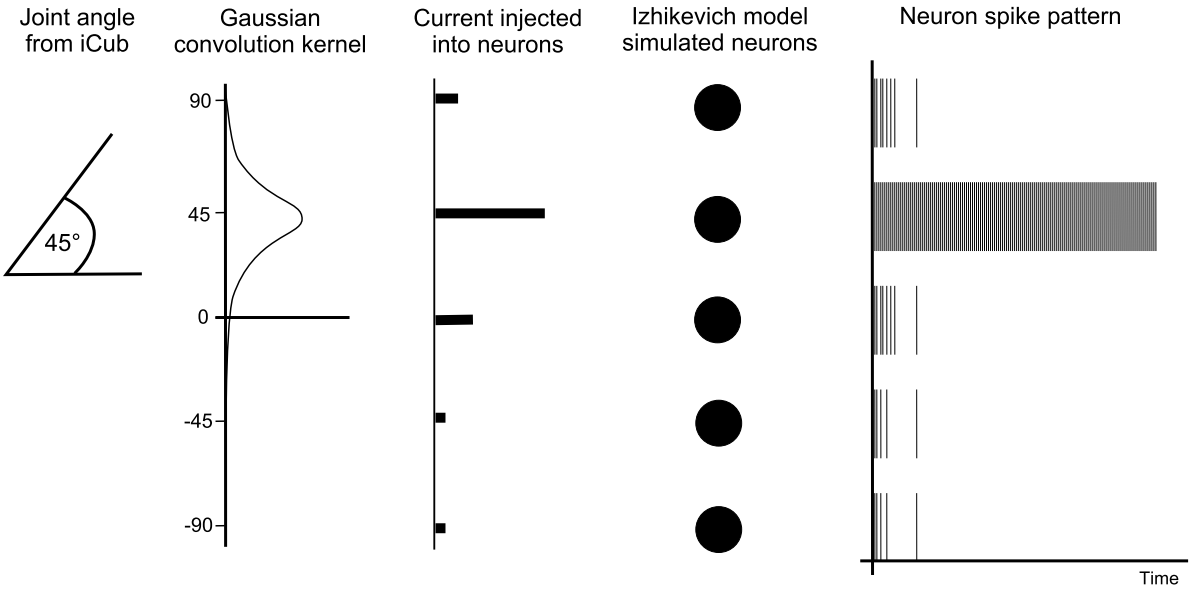


Figure 9. Conversion of joint angle into spikes. The value of the joint position is converted into a Gaussian distribution of current, with a mean equal to the current value and a variance that can be configured to match the measurement noise of the sensors. This current is fed into a population of Izhikevich neurons, whose spiking response can be configured by modifying the neuron parameters. The spiking output of these neurons is passed to the neural simulator.

can be used to reconstruct the joint angle from the population.

4.5. Biological Motor Control

Biological motor control can be roughly divided into two components:

- *Voluntary muscle contraction.* Neurons in the motor cortex and brain stem specify the contraction of groups of muscles to move a part of the body, such as an arm or leg, into a particular position. Circuitry within the spinal cord ensures that antagonists of the contracted muscles are relaxed, and it maintains the body in a particular position.
- *Motor programs in the spinal cord.* Neural circuitry in the spinal cord executes sequential muscle contractions that lead to simple behaviours, such as the withdrawal reflex or running [31]. The brain does not control the order and nature of the individual commands, but sends a trigger to the spinal cord to indicate that a particular movement pattern should be carried out. The spinal cord then executes the individual actions specified by the requested motor program. A good demonstration of this phenomenon is a chicken's ability to continue running when it has been decapitated.

The current version of iSpike focuses on voluntary motor control, both because it would be considerably more complex to create motor programs controlling reflexes and running for the iCub, and because these could be modelled within a neural network

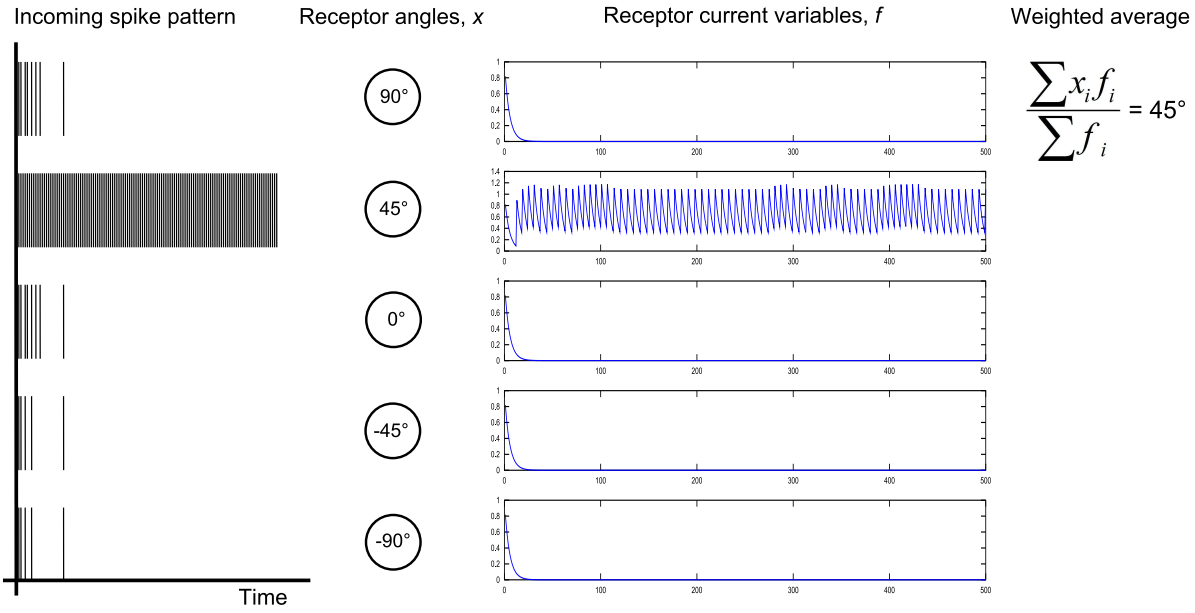


Figure 10. Conversion of spikes into motor commands. The per-joint value is computed by taking the sum of receptor means weighted by the normalised firing rate over a short time window. The resulting joint angle is sent through YARP to the iCub.

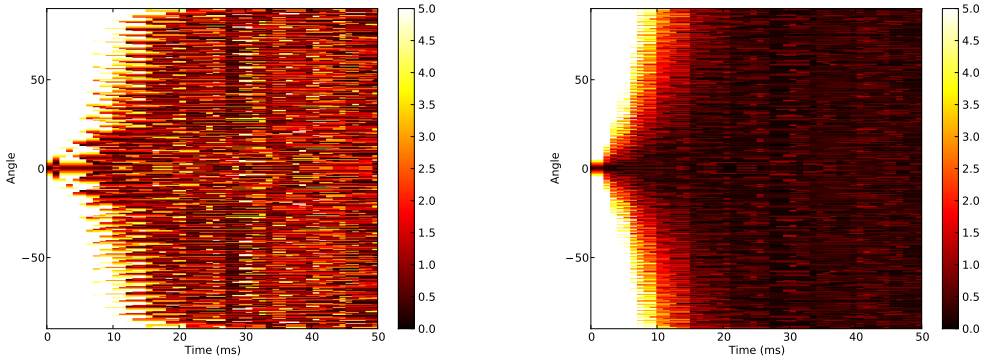
if they were required. Stored motor programs also have the limitation that they are typically finely tuned for a particular body — a circuit that produced running behaviour in the real iCub would be unlikely to work on the virtual iCub. Within the brain most of the voluntary higher level motor control occurs in the primary motor cortex, and there is evidence that a population-based encoding is used to specify movement direction [27].

4.6. iSpike Model of Biological Motor Control

The iCub is controlled by setting joint angles in the robot. In iSpike the spike-to-angle conversion is carried out by an array of receptors that map onto the neurons producing the spike pattern. Each receptor represents a single angle, and it contains a current variable, which is increased by a fixed amount with each observed spike and decays exponentially over time. With continued input from the spike pattern, the receptors receiving spikes more often will have a higher current value and receptors receiving no spikes will eventual decay to zero. The joint angle value is extracted from the receptors by taking the average of the current variables multiplied by the receptor angles. This process is illustrated in Figure 10.

5. Evaluation

An ideal spike conversion library needs to be both accurate and fast. This section discusses the potential data loss introduced by converting between real-valued data and spikes (Section 5.1) and gives some performance measurements of the system



(a) (b)

Figure 11. Encoding and decoding accuracy (absolute error in degrees) for different parts of the angle space, when transitioning from an initial angle of zero. (a) 25 neurons in population; (b) 100 neurons in population.

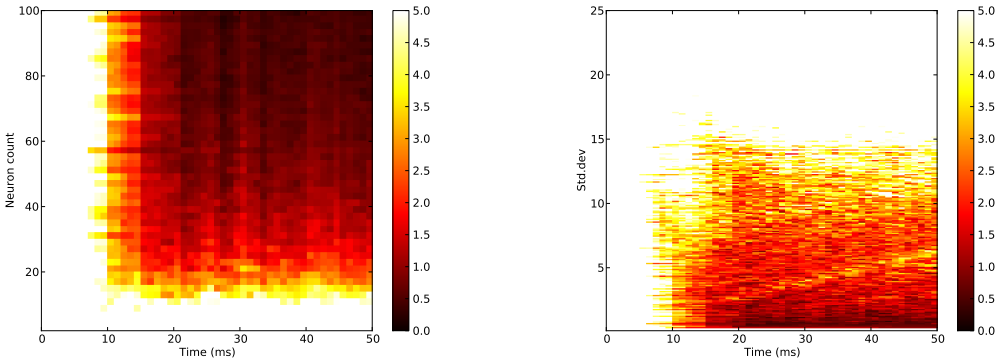
(Section 5.2).

5.1. Joint Angle Conversion Accuracy

The accuracy of the joint angle encoding and decoding was evaluated by converting a series of angles from scalar values into spike patterns and back again. The resulting discrepancy between the initial and final value gives a measure of the accuracy of the spike coding scheme, providing a combined error for both encoding and decoding.

To evaluate whether the whole angle space is encoded with the same quality, we encoded and decoded a large number of angles while keeping a constant number of neurons/receptors (25 or 100) and a fixed normal distribution for the receptive field ($\sigma = 0.5$ neurons). The angle space was sampled at half degree intervals and the combined encoding/decoding error was measured as the system adapted to an instantaneous change from an initial angle of 0 to the new angle. The encoding/decoding error was measured over a period of 50 ms. The results of this experiment is shown in Figure 11. When using a small number of neurons there is some periodicity in the result, which is reduced by increasing the size of the neuron population. The results also show that there is a delay between a change in the input angle and the corresponding change in the decoded angle. When converting from spikes to angle each spike has an effect over a time period (due to the exponential decay) and there is therefore some delay before the new spike pattern dominates. The delay before a stable value is achieved increases with the magnitude of the change in angle, but even large changes in input gives errors of less than five degrees after 10 ms and less than one degree after 15 ms.

The size of the population that is used to encode and decode a value also affects the accuracy, and so we measured the combined encode/decode error for a varying number of neurons between 2 and 100. Figure 12(a) shows the root mean square error (in degrees) over time for a number of runs using different population sizes. The error



(a) (b)

Figure 12. Accuracy of encoding/decoding measured by root mean square error (in terms of angle) for (a) variations of population size, averaged over angles, and (b) variations in perceptive field size, averaged over angles.

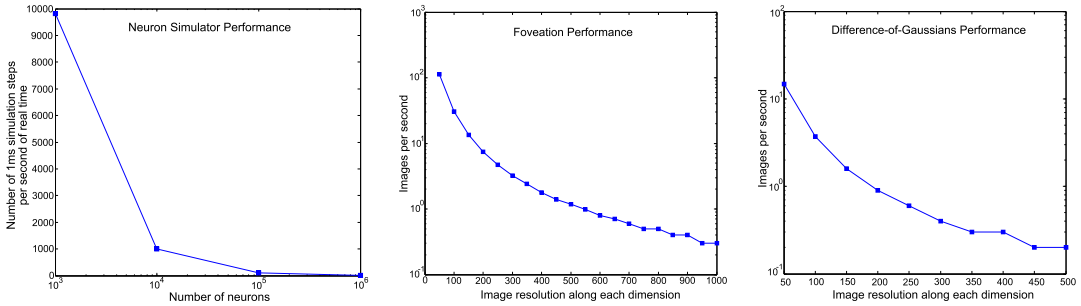
for each time/size pair is the average over 40 runs with transitions between pairs of uniformly random angles in the range $[-45°, 45°]$. The figure shows accuracy improving both with increasing population size and with time. Populations less than ten neurons display error rates in excess of 10 degrees, whereas with population size from around 50, the error is generally less than one degree. For large populations, the error settles at this low value after approximately 20 ms.

When encoding angles into spikes and when decoding spikes into angles, each neuron in the population code has a normal receptive field and variations in the width of this field also affects accuracy. Figure 12(b) shows the root mean square error for a population of 50 neurons with standard deviation (in neurons) ranging from 0.05 to 25, averaged at each time step over 50 runs with different initial and final angles drawn from a uniform distribution. This shows that fairly narrow fields provide better results.

5.2. Performance

The processor-intensive components of iSpike were benchmarked separately to evaluate their impact on the performance. The following tests were carried out using an Intel Core 2 Quad Q6600 CPU, Windows 7 and 4 GB of DDR2 RAM:

- *Neuron simulator.* The conversion of sensory data into spikes depends on Izhikevich neurons, which are modelled in a simple simulator within iSpike. The efficiency of this simulator was tested by creating different sized networks and measuring the average number of 1ms steps that could be simulated in one second of real time. The results shown in Figure 13(a) indicate that a network of 1000 neurons can be simulated 10× faster than real time, whereas a network of 1,000,000 neurons runs 100× slower than real time. In our experiments we found that 100 neurons were more than enough to encode the angle of a particular joint, and so 5300 neurons would be needed to encode the proprioceptive information from the robot's 53



(a) (b) (c)

Figure 13. iCub performance test results. (a) Performance of neuron simulator; (b) Performance of foveation; (c) Performance of difference-of-Gaussians visual filter.

degrees of freedom. The iCub's 320×240 image resolution would require 76,800 neurons to encode, which would run 3× slower than real time.

- *Foveation.* The log polar foveation was run for ten seconds to measure the average number of images that could be foveated at different resolutions. The results in Figure 13(b) show that images with 50×50 resolution can be produced 115 times per second, whereas it takes 3 seconds to produce a single image with a resolution of 1000×1000. At the iCub resolution of 320×240 only four to five images can be produced per second, which is significantly less than the 15 frames per second from the robot's camera.
- *Difference-of-Gaussians visual filters.* The difference-of-Gaussians visual filter was run for ten seconds to measure the average number of images that could be processed at different resolutions. The results shown in Figure 13(c) indicate that it could produce 14.7 opponency maps per second with a 50×50 image, whereas images with a resolution of 500×500 could only be processed once every five seconds. At the 320×240 resolution used by the iCub, the visual filter outputs one opponency map every 1.5 seconds, which is much slower than the 15 frames per second of the iCub's cameras.

5.3. Tests of the Complete System

Tests of the complete system were carried out to establish that a network simulated in SpikeStream/NeMo could receive sensory data from the iCub after it had been encoded by iSpike into spikes, and to check that iSpike could convert spikes from the network into motor output and send this motor output to the iCub. For these tests we created a demonstration network with motor output layers, proprioceptive layers and a visual layer that received information from the R+G- channel. It was found that activity in the motor output layers successfully controlled the robot's joints, and that movements of the robot's joints resulted in appropriate patterns in the proprioceptive and visual layers. Three videos of these experiments are available at http://ispike.sf.net/videos/,

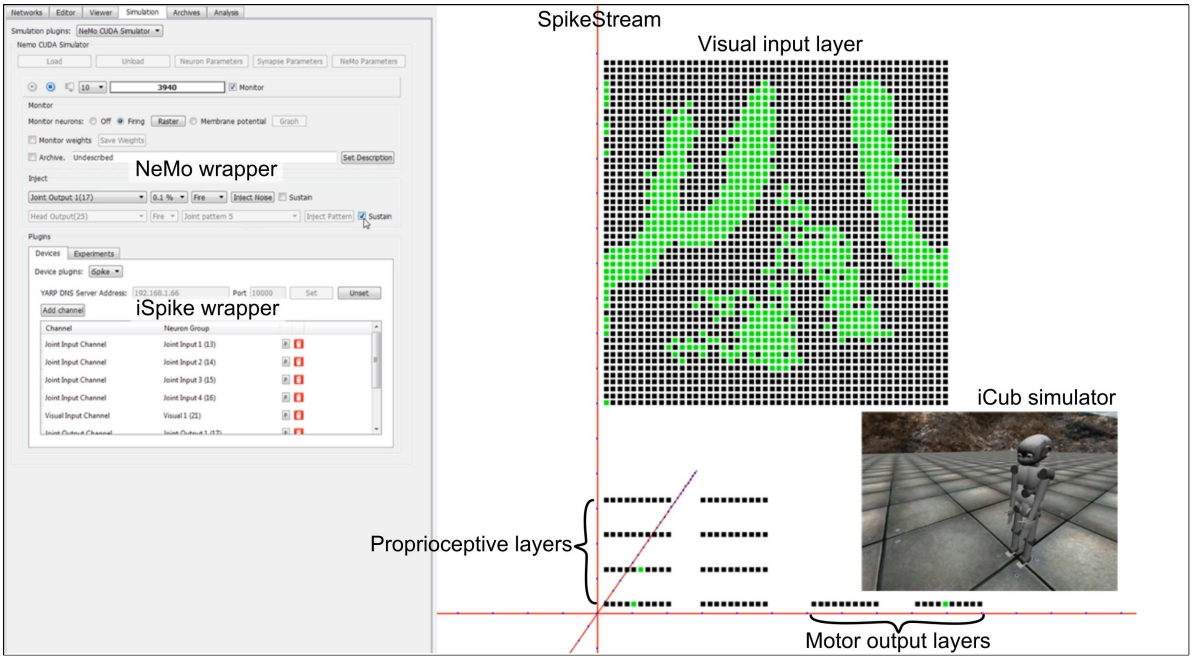


Figure 14. iSpike, SpikeStream, NeMo and the virtual iCub working together.

and a screenshot is shown in Figure 14. ¶

6. Discussion and Future Work

While iSpike's angle conversion was fast enough for real time operation, its visual processing was significantly slower than the iCub's frame rate. In many situations this would not be an issue because the latest image is buffered and spikes from the image are delivered continuously at whatever rate the neural simulator is running. It is also possible to achieve near real time performance by reducing the image resolution. However, if high resolution or fast movements were required, the slow image processing would become a problem, and we are looking at more efficient ways of implementing the foveation and difference-of-Gaussian algorithms. Hardware acceleration on graphics cards could also be used to speed up the visual processing.

A second direction of future work is the extension of the sensory processing to include biologically realistic encoding of audio data into spikes and spike encoding of tactile data. The motor output could be extended to include biologically realistic decoding of spike patterns into audio output, and improvements to the vision could include a motion sensitive magnocellular channel, which would work in a similar way to the silicon retina developed by Delbruck [32]. In future work it would also be worth changing the foveation to Wilson's more biologically realistic model, if the performance issues could be addressed.

¶ These tests were carried out using the iCub simulator because at the time of writing the iCub robot at our institution (Imperial College, London) has been away for refurbishment.

The current version of iSpike should work with any robot that interfaces with YARP, as long as the visual and numerical data is in the same format as the iCub. It would be relatively straightforward to extend iSpike to support other robots and platforms, and it could also be used to convert other types of data to and from spikes. For example, financial data could be encoded into spikes, so that spiking neural networks could be used to identify patterns and predict profitable trades.

The iSpike interface is just a starting point for future work on embodied biologically-inspired spiking neural networks. It will make it easier to build embodied simulated networks that further our knowledge about the brain, and it can help us to create more intelligent robots that use spiking neural networks to process sensory information from their environment and control their bodies in the world. The development of accurate models of sensory encoding and decoding can also improve our understanding of how the brain processes information, which could lead to better brain interfaces and artificial replacements for lost senses. For example, biologically realistic models of the retina could contribute to the development of replacement vision systems that convert camera data into spikes [33, 34]. Biologically realistic decoding of motor output data could help us to develop systems that enable paraplegic patients to control a robot body.

7. Conclusions

This paper has presented iSpike, an interface that encodes sensory information from a robot into spikes and converts spike patterns into motor commands that are sent to the robot. iSpike has been developed to enable hardware-accelerated neural simulators, such as NeMo, to be used in conjunction with the iCub and other robots, and it is compatible with any system that can interface with a C++ library. Future plans include improving the performance, extending the support for sensory and motor encoding and decoding, and using iSpike in conjunction with SpikeStream, NeMo and the iCub to model aspects of the brain and explore how intelligent systems can be developed using spiking neural networks.

Acknowledgements

This work was supported by EPSRC grant EP/F033516/1.

References

[1] W. Maass and C.M. Bishop, editors. *Pulsed Neural Networks*. MIT Press, Cambridge, Mass.; London, 1999.

[2] A. K. Fidjeland and M. P. Shanahan. Accelerated simulation of spiking neural networks using GPUs. In *Proc. IEEE International Joint Conference on Neural Networks*, July 2010.

[3] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, 22(5-6):791–800, 2009.

[4] E. M. Izhikevich and G. M. Edelman. Large-scale model of mammalian thalamocortical systems. *Proc Natl Acad Sci U S A*, 105(9):3593–8, 2008.

[5] J. L. Krichmar, D. A. Nitz, J. A. Gally, and G. M. Edelman. Characterizing functional hippocampal pathways in a brain-based device as it solves a spatial memory task. *Proc Natl Acad Sci U S A*, 102(6):2111–6, 2005.

[6] A. Clark. *Supersizing the mind : embodiment, action, and cognitive extension*. Oxford University Press, New York ; Oxford, 2008.

[7] J. K. O'Regan and A. Noe. A sensorimotor account of vision and visual consciousness. *Behav Brain Sci*, 24(5):939–73; discussion 973–1031, 2001.

[8] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori. The iCub humanoid robot: an open platform for research in embodied cognition. *Proc. Workshop on Performance Metrics for Intelligent Systems*, 2008.

[9] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Trans Neural Netw*, 14(6):1569–72, 2003.

[10] Y. Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In H. Araki, editor, *Mathematical Problems in Theoretical Physics*, volume 39 of *Lecture Notes in Physics, Berlin Springer Verlag*, pages 420–422, 1975.

[11] S. Song, K. D. Miller, and L. F. Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926, 2000.

[12] D. Gamez and I. Aleksander. Accuracy and performance of the state-based F and liveliness measures of information integration. *Consciousness and Cognition*, In press, 2011.

[13] D. Gamez, R. Newcombe, O. Holland, and R. Knight. Two simulation tools for biologically inspired virtual robotics. In *Proceedings of the IEEE 5th Chapter Conference on Advances in Cybernetic Systems*, pages 85–90, 2006.

[14] D. Gamez. Information integration based predictions about the conscious states of a spiking neural network. *Consciousness and Cognition*, 19(1):294–310, 2010.

[15] A. Bouganis and M. Shanahan. Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity. In *Proc. IEEE International Joint Conference on Neural Networks*, pages 4104–4111, 2010.

[16] S. Deneve, P. E. Latham, and A. Pouget. Reading population codes: a neural implementation of ideal observers. *Nat Neurosci*, 2(8):740–5, 1999.

[17] W. Bialek, F. Rieke, R. R. de Ruyter van Steveninck, and D. Warland. Reading a neural code. *Science*, 252(5014):1854–7, 1991.

[18] A. Novellino, P. D'Angelo, L. Cozzi, M. Chiappalone, V. Sanguineti, and S. Martinoia. Connecting neurons to a mobile robot: an in vitro bidirectional neural interface. *Intell. Neuroscience*, 2007:2–2, January 2007.

[19] T. Masquelier and S.J. Thorpe. Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Comput Biol*, 3(2):e31, 02 2007.

[20] A.G. Andreou and K.A. Boahen. A 48,000 pixel silicon retina in current-mode subthreshold cmos. In *37th Midwest Symposium on Circuits and Systems*, pages 97–102, 1994.

[21] A. Linares-Barranco, F. Gomez-Rodriguez, A. Jimenez-Fernandez, T. Delbruck, and P. Lichtensteiner. Using FPGA for visuo-motor control with a silicon retina and a humanoid robot. In *Proc. IEEE. Int. Symposium on Circuits and Systems*, pages 1192–1195, may 2007.

[22] E. D. Adrian. The impulses produced by sensory nerve endings: Part i. *J Physiol*, 61(1):49–72, 1926.

[23] D. Bendor and X. Wang. Differential neural coding of acoustic flutter within primate auditory cortex. *Nature Neuroscience*, 10(6):763–771, April 2007.

[24] J. Gautrais and S. Thorpe. Rate coding versus temporal order coding: a theoretical approach. *Bio Systems*, 48(1-3):57–65, 1998.

[25] T. Gollisch and M. Meister. Rapid neural coding in the retina with relative spike latencies. *Science*, 319(5866):1108–11, 2008.

[26] R. Van Rullen and S.J. Thorpe. Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex. *Neural Comput.*, 13:1255–1283, June 2001.

[27] A. P. Georgopoulos, J. Ashe, N. Smyrnis, and M. Taira. The motor cortex and the coding of force. *Science*, 256(5064):1692–5, 1992.

[28] M.F. Bear, B.W. Connors, and M.A. Paradiso. *Neuroscience : exploring the brain*. Lippincott Williams & Wilkins, Baltimore, Md., 2nd ed. edition, 2001.

[29] M. Bolduc and M.D. Levine. A review of biologically motivated space-variant data reduction models for robotic vision. *Comput. Vis. Image Underst.*, 69:170–184, February 1998.

[30] C. Enroth-Cugell and J.G. Robson. The contrast sensitivity of retinal ganglion cells of the cat. *The Journal of Physiology*, 187(3):517–552, 1966.

[31] S. Grillner, J. Hellgren, A. Menard, K. Saitoh, and M. A. Wikstrom. Mechanisms for selection of basic motor programs–roles for the striatum and pallidum. *Trends Neurosci*, 28(7):364–70, 2005.

[32] T. Delbrck. Silicon retina with correlation-based, velocity-tuned pixels. *IEEE Transactions on Neural Networks*, 4(3):529541, 1993.

[33] M. Javaheri, D. S. Hahn, R. R. Lakhanpal, J. D. Weiland, and M. S. Humayun. Retinal prostheses for the blind. *Ann Acad Med Singapore*, 35(3):137–44, 2006.

[34] J. F. Rizzo. Update on retinal prosthetic research: the boston retinal implant project. *J Neuroophthalmol*, 31(2):160–8, 2011.