Pattern Recognition Letters xxx (2013) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm

Salvador Dura-Bernal [a,*], George L. Chadderdon [a], Samuel A. Neymotin [b], Joseph T. Francis [a,e,f,g], William W. Lytton [a,c,d,e,f,g]

[a] Department of Physiology and Pharmacology, State University of New York Downstate, Brooklyn, NY, USA
[b] Department of Neurobiology, Yale University School of Medicine, New Haven, CT, USA
[c] Department of Neurology, State University New York Downstate, Brooklyn, NY, USA
[d] Department of Neurology, Kings County Hospital, Brooklyn, NY, USA
[e] Joint Program in Biomedical Engineering at Polytechnic Institute of New York University and State University of New York Downstate, Brooklyn, NY, USA
[f] Program in Neural and Behavioral Science at State University of New York Downstate, Brooklyn, NY, USA
[g] The Robert F. Furchgott Center for Neural & Behavioral Science, State University of New York Downstate Medical Center, Brooklyn, NY, USA

ARTICLE INFO

Article history:
Available online xxxx

Communicated by Luis Gomez Deniz

Keywords:
Real-time interface
Robotic arm
Sensorimotor cortex
Biomimetic model

ABSTRACT

Brain–machine interfaces can greatly improve the performance of prosthetics. Utilizing biomimetic neuronal modeling in brain machine interfaces (BMI) offers the possibility of providing naturalistic motor-control algorithms for control of a robotic limb. This will allow finer control of a robot, while also giving us new tools to better understand the brain's use of electrical signals. However, the biomimetic approach presents challenges in integrating technologies across multiple hardware and software platforms, so that the different components can communicate in real-time. We present the first steps in an ongoing effort to integrate a biomimetic spiking neuronal model of motor learning with a robotic arm. The biomimetic model (BMM) was used to drive a simple kinematic two-joint virtual arm in a motor task requiring trial-and-error convergence on a single target. We utilized the output of this model in real time to drive mirroring motion of a Barrett Technology WAM robotic arm through a user datagram protocol (UDP) interface. The robotic arm sent back information on its joint positions, which was then used by a visualization tool on the remote computer to display a realistic 3D virtual model of the moving robotic arm in real time. This work paves the way towards a full closed-loop biomimetic brain-effector system that can be incorporated in a neural decoder for prosthetic control, to be used as a platform for developing biomimetic learning algorithms for controlling real-time devices.

1. Introduction

Understanding the human brain has become one of the great challenges of this century (Lytton et al., 2012). It has the potential to significantly benefit human health by providing tools to treat neural disease and repair neural damage. Greater understanding will also revolutionize our interactions with the world through development of BMIs for motor control that enable patients with central nervous system injuries or lost limbs to regain autonomy. Neuroscience research is also beginning to make progress towards understanding how neurons encode information, and how the complex dynamical interactions within and among neuronal networks lead to learning, and produce sensorimotor coordination and motor control. With this knowledge, we can begin to use computers both to decode brain information and to autonomously produce brain-like signals to control prosthetic devices, or to control a real arm (Carmena et al., 2003).

A key problem with simulation technology is that one can not be sure whether something critical, known or unknown, has been left out. This concern suggests an approach whereby we embed biomimetic neuronal networks in the actual physical world in which the real brain is embedded. We thereby both take advantage of the benefits of physicality and ensure that our systems are not omitting some factor that is required for actual functionality. Physicality provides a form of memory, since the arm is actually located in space and will later be in the same location unless operated on by external forces such as gravity. It also provides inertia, which is memory for the 1st derivative of position. However, these mnemonic attributes, features under what set of circumstances, can be limitations (bugs) when the system is presented with other operational requirements.

* Corresponding author at: Department of Physiology, State University of New York Downstate, Brooklyn, NY, USA. Tel.: +1 917 446 2747; fax: +1 815 642 4019.
E-mail address: salvadordura@gmail.com (S. Dura-Bernal).

0167-8655/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.patrec.2013.05.019

The physical world also plays an essential role in learning and behavior (Almassy et al., 1998; Edelman, 2006; Krichmar and Edelman, 2005; Lungarella and Sporns, 2006; Webb, 2000). For example, the selection hypothesis emerges from embodiment: the environment can select neuronal dynamics that are suitable for producing desired behaviors through the agency of a limb or other effector (Edelman, 1987). Embodiment can be used to make predictions for how changes will occur during the perception–action–reward–cycle (Mahmoudi and Sanchez, 2011), as the embedding of a system in the environment provides adaptation opportunities. Further levels of embedding are achieved by the development of hybrid systems that use *co-adapting* symbiotic relations between brain (the prosthetics-user) and artificial agents (biomimetic systems and smart prosthetic devices) and among these artificial agents. The co-adaptions occur as the system moves towards common goals, e.g., reaching for an object (Sanchez et al., 2012). In this context, one can readily see the advantages of biomimesis: the biomimetic system is a partial model of the natural agent's brain processes that facilitates the transfer of neural encodings without explicit decoding, providing representations based on those of the brain and then serving as intermediary between brain and devices. We note that co-adaptation is further extended to include the environment itself, as when we redesign objects so as to make them easier to manipulate with artificial limbs, or easier to navigate for individuals in wheelchairs.

Biomimetic brain models (BMMs) are able to replicate many experimental paradigms, such as sensorimotor learning experiments (Chadderdon et al., 2012; Neymotin et al., 2013) or cellular microstimulation (Kerr et al., 2012). They are also able to accurately reproduce physiological properties observed *in vivo*, including firing rates, stimulus-induced modulations and local field potentials (Neymotin et al., 2011). The system presented here will be extended into a framework to link real-time electrophysiological recordings with supercomputers that run the BMMs, and thence to control of prosthetic devices. This co-adaptive brain–machine interface extends the classical BMI paradigm by engaging both subject and computer model in synergistic learning.

Major challenges in assembling a system that incorporates a BMM into the neural decoder/smart prosthetic data stream are (1) getting the components to communicate with one another, and (2) achieving this communication in real-time. There are a plethora of different systems for acquiring electrophysiological data from animal or human subjects (Buzsáki, 2004), a number of potential neuronal simulators (biomimetic and state-space) that might be assembled together to provide the brain model (Brette et al., 2007), and a wide array of potential prosthetic links with different physical characteristics that require different control strategies. Software components within this data stream may run on, for example, machines running MATLAB under Windows, or on machines running Python or C++ code under Linux. A networking framework needs to be developed that can not only permit messages to be passed between these disparate environments and hardware platforms, but to do so in a timely fashion so that the prosthetic-using subject does not perceive a disruptive lag in the performance of a prosthetic limb.

In this paper, we address the initial problems of developing the larger real-time co-adaptive BMI system. We begin with the design of inter-system communications between the BMM and the prosthetic limb, leading towards a real-time interface between a model of sensorimotor cortex and a robotic arm (Barrett Technology's Whole Arm Manipulator – WAM (Barrett Technology Inc., 2012a)). We provide the NEURON-based BMM and the robotic arm, each running on a separate Linux-based machine. Our implementation of the real-time interface then provides a Python application that forwards data from the BMM to the WAM arm and passes robot arm position information back to a display window.

2. Methods

We used a BMM previously developed within the NEURON simulation environment using an extended synaptic functionality (Lytton et al., 2008). The original model drove a simple kinematic two-joint virtual arm in a motor task requiring trial-and-error convergence on a single target. We utilized the output of this model to drive mirroring motion of a Barrett Technology WAM robotic arm in real time through a UDP interface. Additionally, the position information from the robot arm was fed back into the model. This feedback then drived a virtual WAM robotic arm developed within V-REP, an open-source robot simulation environment (Freese et al., 2010). This permitted the remote computer to display a representation of the moving robotic arm in real time.

A set of interfaces was designed to provide communication between the NEURON model and the WAM robotic arm. The external PC ran NEURON code, which called a set of Python functions. These interfaced with the WAM arm via UDP communication, and with the V-REP virtual robotic arm via a Python application programming interface (API). The robot code ran on the WAM internal PC, utilizing functions developed in the C++ Libbarrett library (Barrett Technology Inc., 2012b) to provide interface with the NEURON simulation via UDP communication.

The BMM, the robotic arm, the virtual robotic arm and the network interface between these components are described below in more detail.

2.1. NEURON-based biomimetic brain model

2.1.1. Overview of the system and background

For our BMM, we used the spiking neuronal model of sensorimotor cortex previously developed (Fig. 1) (Chadderdon et al., 2012; Neymotin et al., 2013). This model had been trained to perform a simple movement task: rotating a two-joint virtual arm to a target by learning an appropriate mapping between the neural populations through reinforcement learning mechanisms. From

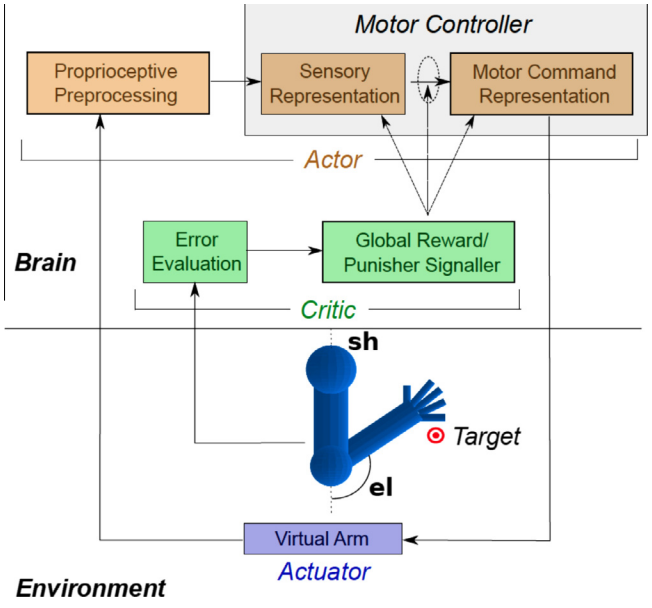

Fig. 1. Overview of the biomimetic model of sensorimotor cortex. A 2-joint virtual arm is trained to reach towards a target using reinforcement learning. A proprioceptive preprocessing area feeds arm information to the sensory region, which is connected to the motor units that drive the muscles to change the joint angles. The Actor is trained by the Critic which evaluates error and provides a global reward or punishment signal. Reproduced from Chadderdon et al. (2012) with permission.

Please cite this article in press as: Dura-Bernal, S., et al. Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm. Pattern Recognition Lett. (2013), http://dx.doi.org/10.1016/j.patrec.2013.05.019

the pattern recognition perspective, our model can be interpreted as a classifier that learns to generate the required output (motor command) based on the input patterns (joint positions and target). Although it is possible that machine learning techniques, such as gradient-based deterministic algorithms, can efficiently solve this problem, our model has the advantage of being based on realistic neuronal learning mechanisms and dynamics. This is a key aspect for the long-term goals of this work, which are described in *Introduction* and *Discussion*.

Reinforcement learning is used as a mechanism for learning. The essence of this learning mechanism was summarized over 100 years ago in Thorndike's Law of Effect (Thorndike, 1911): stimulus–response mappings are strengthened by global reward and weakened by global punishment. Reinforcement learning methods (Sutton, 1998) have been used extensively in machine learning and offer an advantage over teacher-supervised learning methods in that they do not require a known desired output representation to match against the models current (behavioral) output. However, unlike unsupervised learning methods, they do offer some feedback regarding fitness of the behavior.

A further framework for explaining motor reinforcement learning is the perception–action–reward–cycle. The learning system is divided into an actor, mapping perceptions to actions (P to A), and a critic providing reward and punishment feedback to the actor (Chadderdon et al., 2009; Joel et al., 2002). To utilize this scheme, the naive actor must produce some actions that can be critiqued. This is the role of motor babble (random movements), produced in our model via noise.

One challenge in the learning of actor/critic RL systems is the distal reward or credit assignment problem (Izhikevich, 2007): reinforcers are delivered after the behavior is complete, after synaptic and neuronal activations leading up to the output are no longer active. A synaptic eligibility trace is typically used to solve this problem: neuron synapses, where learning occurs, are tagged to receive a credit or blame signal that arrives later (Sutton, 1998). In the current model we trained synaptic weights between spiking units using global reward and punisher signals.

2.1.2. Model implementation

Individual neurons were modeled as rule-based dynamical units with several key features found in real neurons, including adaptation, bursting, depolarization blockade, and voltage-sensitive NMDA conductance (Lytton and Stewart, 2005; Lytton and Stewart, 2006; Lytton and Omurtag, 2007; Lytton et al., 2008; Lytton et al., 2008; Neymotin et al., 2011b; Kerr et al., 2012). The model consisted of 384 excitatory and 128 inhibitory cells, each with three types of synaptic inputs commonly found in cortex (AMPA, NMDA and $GABA_A$). Cells were connected probabilistically (only a subset of all possible connections were made) with connection densities and initial synaptic weights varying depending on pre- and post-synaptic cell types. The cells were arranged into three different populations with realistic and anatomical properties. Input to the sensory cells was provided by 96 position (P) cells, representing the four muscle lengths (flexor and extensor muscles for each joint). The sensory population, which received spiking input from position cells, included 192 excitatory sensory (ES) cells, 44 fast spiking sensory interneurons (IS), and 20 low-threshold spiking sensory interneurons (ILS). The motor population, which received spiking input from sensory cells, consisted of 192 excitatory motor (EM), 44 fast spiking motor interneurons (IM), and 20 low-threshold spiking motor interneurons (ILM). The EM population was divided into four 24-cell subpopulations which controlled each of the four muscles. Joint positions were updated at 50 ms intervals, based on extensor and flexor EM spike counts.

2.1.3. Model training and evaluation

During training, plasticity was present at three sites in the sensorimotor network: E→E recurrent connections in both S and M areas; bidirectional E→E connections between S and M areas; and local E→I connections within S and M areas; where E refers to excitatory, I to inhibitory, S to sensory and M to motor. The *Critic*, a global reinforcement signal, was driven by the first derivative of error between position and target during two successive time points (reward for decrease; punishment for increase). We used a spike-timing-dependent rule to trigger eligibility traces to solve the credit assignment problem (Izhikevich, 2007). When reward or punishment was delivered, eligibility-tagged synapses were potentiated (long-term potentiation LTP), or depressed (long-term depression LTD), correspondingly. Reinforcement occurred at the time of joint position updating (every 50 ms). The system was able to learn the appropriate mappings between neural populations required for target acquisition.

Model evaluation involved over 2000 simulations, each with a duration of 15 s, of previously trained networks. Multiple networks were produced using five different random wirings and five different pseudorandom driving-input streams in order to ensure that positive, or negative, results, were not simply the result of particular wiring or input settings which biased the learning. We additionally assessed five different targets, using 16 different initial arm positions for each. Initial training required 400,000 simulations (five random wirings, five input streams, five targets, 16 initial arm positions, 200 reaches from each position), each with a duration of 15 s.

Learning produced multiple alterations in the activity map and in consequent network dynamics. These changes included a 3-fold increase in excitatory weight gains between the different populations after training to reach the two-degree-of-freedom virtual arm towards a single target. Several of the populations also showed enhanced synchrony, visible as vertical stripes in the raster (Fig. 2). P cells tuned to a specific subset of joint angles (green: shoulder; orange: elbow), demonstrate the arm's position and follow a trajectory in reaching towards the target. Model performance, measured as a fraction of trials where joint positions ended within 10 degrees of target location, was substantially augmented with training (Chadderdon et al., 2012; Neymotin et al., 2013). After training and across targets, the average success ratio was 73% and 68% for shoulder and elbow angles, respectively, compared

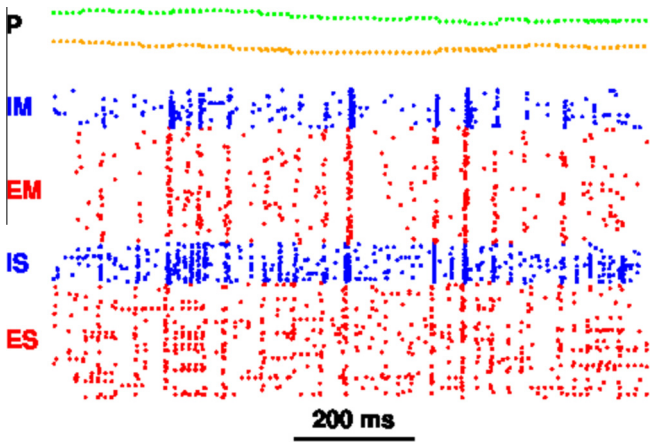

Fig. 2. Raster plot of BMM after training. Raster plot, where blue (red) dots are spikes in inhibitory (excitatory) cells; ES, IS, ILS, EM, IM, ILM (E excitatory; I inhibitory fast-spiking; IL inhibitory low-threshold spiking interneurons; S higher-order sensory; M motor; P position sensory; green: shoulder; orange: elbow). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Please cite this article in press as: Dura-Bernal, S., et al. Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm. Pattern Recognition Lett. (2013), http://dx.doi.org/10.1016/j.patrec.2013.05.019

to only 19% and 24% for the naive networks. Success was calculated for naive and trained networks from identical initial conditions (starting position and random inputs).

The model of sensorimotor cortex was implemented in NEURON 7.2 (Carnevale and Hines, 2006) for Linux and is available on ModelDB (Peterson et al., 1996) (https://senselab.med.yale.edu/modeldb).

2.2. WAM robotic arm

Barrett Technology has developed the Whole Arm Manipulator (WAM) as a compact, low weight, low power consumption robotic arm providing smooth and precise joint motion. These features make it well suited for robotics control research. We used 2 degrees of freedom (shoulder flexion/extension, elbow flexion/extension) out of the 7 available in our WAM configuration. The WAM internal PC, embedded in the base of the WAM arm, controlled the robot arm motors through a 2-wire differential serial bus (CAN bus) that provided digital communication at 1 Mbps. The control loop, which runs by default at 500 Hz, sent motor torques from the WAM internal PC to the WAM arm motors and sent back the motor positions from the motors to the internal PC (see Fig. 3). An open-source C++ library, Libbarrett, provided by Barret Technology, included high-level functions to control the WAM arm from the internal PC. When a command was sent from the internal PC, it was translated into CAN signals and addressed to the appropriate motors. The WAM included a small router attached to the outside of the robot's base that allows an external PC to connect to the internal WAM PC, to both remotely run code in the internal PC to provide 2-way communication in real-time between internal and external PCs. In the interface described in this paper, we employed this mechanism to control the robot arm from an external PC and receive information on the joint positions in real time.

2.3. V-rep virtual robotic arm

V-REP is a new open-source robot simulation platform that allows creation of detailed and realistic simulations of robots and experimentation with them in virtual worlds. It included communication APIs with common programming languages (Python, Matlab (MathWorks, Inc., 2012), C++, and others) that allowed control of a virtual robot from any external application and receiving of feedback from the simulator. Additionally, the virtual robot output can be used to control a real robot in real-time. The simulation environment provided easy-to-use and powerful tools, including a library of 3D objects, robots and devices, several types of sensors (vision, tactile, etc.) that can feed data to the robot, and the ability to plot graphs of any of the variables in the simulation.

We added a model of the Barrett WAM arm to the V-REP simulator, appending to an existing model of the Barrett hand. This virtualization characterized over 20 physical, kinematic, and dynamical parameters of the real device. It provided inverse and forward kinematics calculations and 2 physics engines for dynamics calculations, enabling simulations of real-world physics and object interactions, including collisions and grasping. The WAM virtualization was used for two purposes: 1. replace the real robot arm during experimentation (tested by sending BMM output to both the real and virtual WAM in parallel); 2. provide real time 3D visualization of the arm movements.

2.4. Biomimetic brain model and network communication on external PC

The Python interface functions are used to set up the communication sockets during initialization and to periodically send and receive joint angles to and from the WAM PC. In the arm mirroring

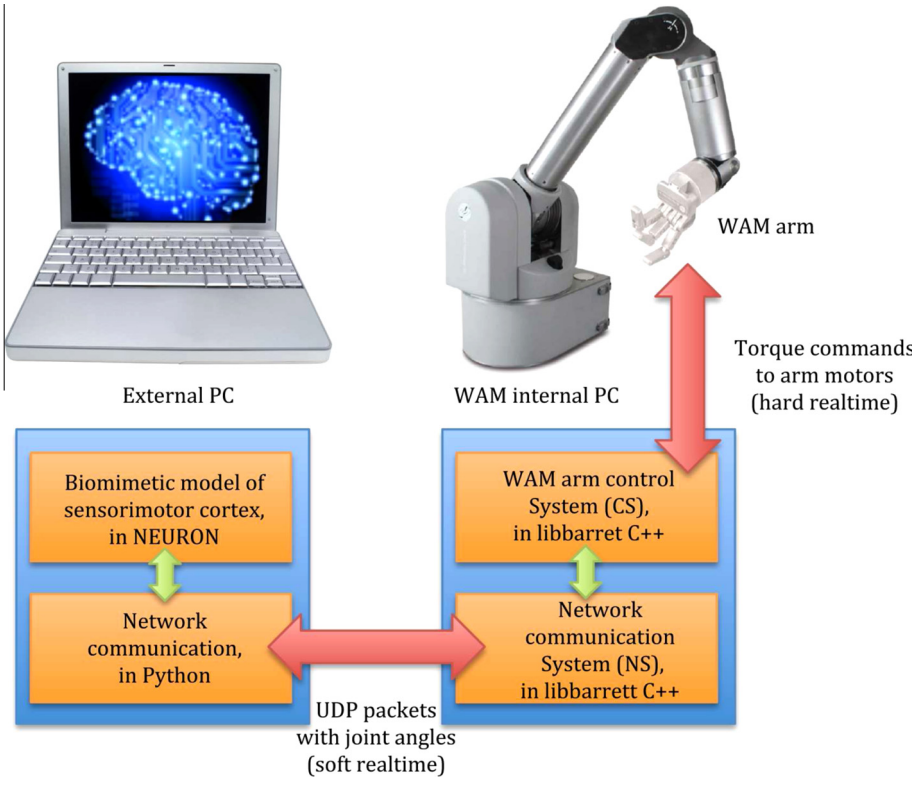

Fig. 3. Diagram of interface between BMM and WAM robot arm. The external PC running the BMM sends the desired joint angles to the WAM internal PC, which converts them into joint torques that are sent to the WAM arm. The process also occurs in the opposite direction, where joint angles are fed from the WAM arm to the WAM internal PC and then via UDP to the external PC back into the BMM.

Please cite this article in press as: Dura-Bernal, S., et al. Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm. Pattern Recognition Lett. (2013), http://dx.doi.org/10.1016/j.patrec.2013.05.019

scheme, we transpose the single horizontal plane of the NEURON-simulated 2D arm into the vertical plane of the WAM arm.

The Python interface functions are loaded into the NEURON simulation environment and called directly from NEURON. Python being one of the two scripting languages used in the NEURON simulator (Carnevale and Hines, 2006). The communication function worked by opening a UDP socket between the external PC and the WAM internal PC. It then converted the output NEURON joint angles into UDP packets, and forwarded these through the socket to the robot arm. Finally, it received the joint position information coming back from the robot arm via the same UDP socket and saved it to a data file (Fig. 3). The current simulation model performed these operations at 50 ms simulated-time intervals (currently 150 ms real-time). UDP packets were time-stamped by both simulated time and by real-time, both measured from simulation start. The rate of incoming packets was, however, much higher, as they originated in the WAM PC, which operates strictly at 500 Hz (2 ms intervals). For this reason, when the buffer was read, all incoming packets were discarded, except the last one which contained the most up-to-date information from the WAM.

The joint angles received from the robotic arm were available to the BMM. In future, these will be provided as feedback information to the model during learning. Currently, we used these angles only to provide real-time visualization of the movements on the external PC using the WAM virtualization in V-REP. This required development of the real-time interface between the BMM and the virtual WAM arm using a Python API provided by V-REP. The interface worked in a similar fashion to that of the real WAM, sending packets containing the shoulder and elbow angles. However, in this case, the data was sent locally within the same computer.

2.5. Robot control and network communication on WAM internal PC

The open-source Libbarrett C++ library was the primary code base used to control and operate the WAM arm from the WAM internal PC. It connected functional blocks called Systems, which are small virtual machines that process information from inputs to outputs. The output of a System can be connected to the input of one or several Systems in order to obtain the desired functionality. This is similar to the process of linkage used in MathWorks' Simulink (MathWorks, Inc., 2012).

We implemented two Systems: a Network System (NS), to handle UDP communication, and a Control System (CS), for controlling and interacting directly with the robot arm. The NS established the UDP socket, received UDP packets from the external PC containing the desired joint angles, and sent back to the external PC the joint angles of the WAM arm received from the CS. The CS received joint angles from the NS, converted them to joint torques, and sent these to the robot motors. Additionally, the CS fed back the joint angles from the robot motors to the NS (Fig. 3).

Communication between CS and robot arm motors must run in *hard* real-time, operating at a fixed frequency of 500 Hz. In each cycle, joint torques were sent to the motors and joint angles were received back. Any delay in this process resulted in a fault, causing the robot to shut down. By contrast, communication between NS and external PC can run in *soft* real-time: the delays for each cycle may vary significantly due to the local-area-network properties. Additionally, a small percentage of packets may be lost. In order to prevent the soft real-time NS from adversely affecting the hard real-time CS, the former was implemented using non-blocking UDP communication. Additionally, in cases where UDP packets were not received, the NS continued to output the joint angles from the last packet it had received from the Python interface. This ensured that the CS received a constant input of joint angles even when the NS lagged. This also eliminated the need for sending replicated joint angle packets from the Python interface, and led to smoother trajectories.

Currently the arm is controlled using positional, rather than velocity or force information. We therefore needed a way of adjusting the speed or transition smoothness between the joint angles sent from the BMM simulation to the WAM arm. In collaboration with the Barrett software team, a new class was added to the Libbarrett library in order to permit the setting of a fixed speed for each joint, in radians per cycle, so as to make it possible to directly use positional information to control the WAM arm. This new method was employed to transform the output of the NS before it was fed to the CS to control the arm. In this way, we generated a smooth trajectory that could be safely followed by the WAM arm. The speed parameters used were 0.0018 and 0.00275 radians per cycle, for the shoulder and elbow joints, respectively.

Internally, the WAM arm is controlled using a proportional-integral-derivative (PID) controller that gathers positional error feedback and attempts to minimize that error by adjusting the motor torque. More specifically, the PID operates proportionally to the error (k_p), the integral of the error (k_i), and the derivative of the error (k_d). We tuned these parameters to our system, based on the rate of incoming packets and on the joint speed parameters. The resulting values were $k_p = 4375, k_i = 5, k_d = 10$ for the shoulder joint, and $k_p = 875, k_i = 0.5, k_d = 1$ for the elbow joint.

3. Results

3.1. Experimental set-up

The BMM ran on an external PC (laptop with 2.66 GHz Intel Core i7) and sent joint angles via wireless LAN to the WAM internal PC controlling the WAM arm. The resultant WAM joint angles were fed back to the BMM via wireless LAN and forwarded to the WAM virtual virtualization, running under V-REP on the same external PC (Fig. 4). The joint angles are passed through the BMM software so that they can later be used as inputs for the learning loop. The left window in Fig. 4 laptop screen shows the textual output of the BMM, while the right window of the laptop shows the V-REP virtual WAM arm. A demonstrative video is available at: www.neurosimlab.org/salvadord/PRL_WAMarm_video.m4v.

Fig. 4. Screen capture of video taken during experiment. The BMM, running in the laptop (left window), controls in real time both the real WAM arm, via LAN network, and the V-REP virtual WAM arm, running in the same laptop (right window).

Please cite this article in press as: Dura-Bernal, S., et al. Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm. Pattern Recognition Lett. (2013), http://dx.doi.org/10.1016/j.patrec.2013.05.019

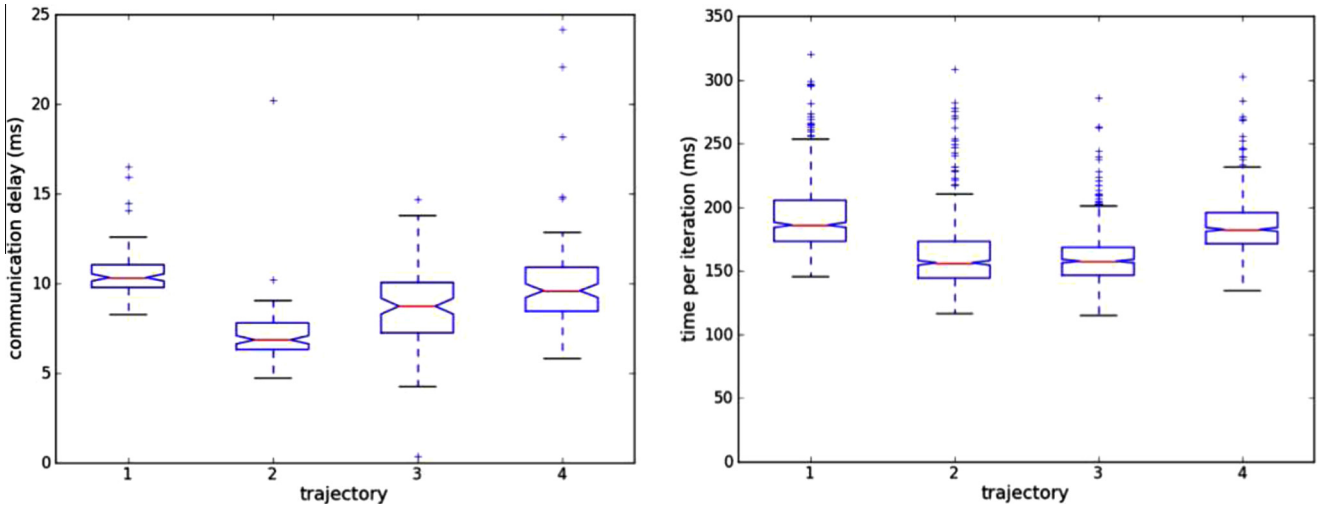

Fig. 5. Communication delay and time per iteration. (left) Box-and-whisker plot of the communication delay of packets sent from the BMM to the WAM via LAN. (right) Box-and-whisker plot of the time per iteration in the BMM.

We tested four different trajectories generated by the BMM using different targets, starting positions and network wirings. Each simulation ran for 30 s of simulated time, which required approximately 100 s of real time. The difference in times was due to the slowness of the current BMM, rather than to delays in the communication loops. To evaluate interface performance, we synchronized the PC clocks using Network Time Protocol (NTP), and recorded sent and received time-stamped joint angles both at BMM and WAM, as well as time-stamped data packets.

3.2. Communication delay

Mean communication delay over all trajectories was 9.16 ms, well within the bounds required for real-time performance (Fig. 5). The communication delay of packets sent from the external PC (BMM) to the WAM PC, over the first 100 packets, was comparable for all four trajectories (left panel). Mean duration of each iteration over all trajectories was 174.76 ms, a time that included calculating the shoulder and elbow angles based on the model

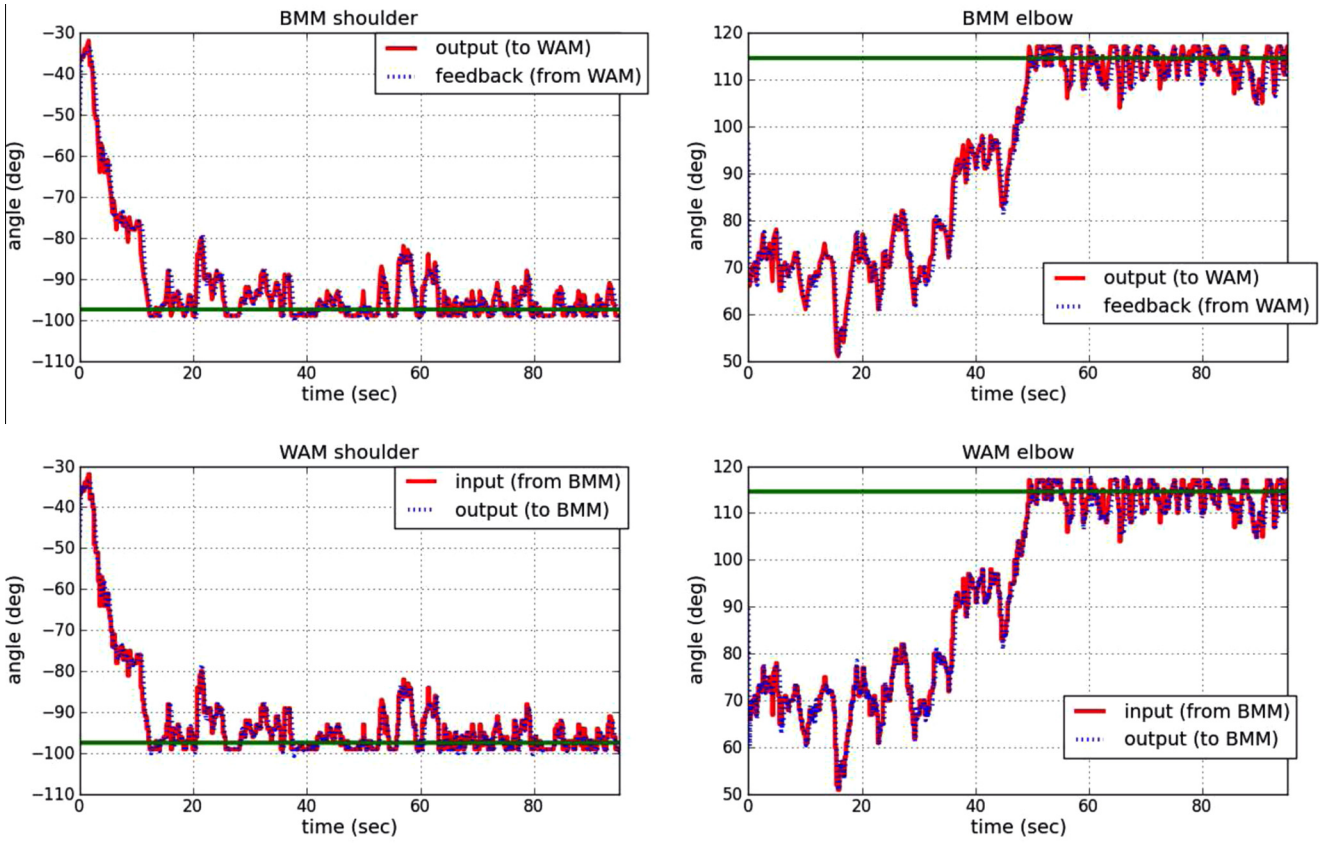

Fig. 6. Comparison of joint angles over a 95-s trajectory. The arm shoulder (left) and elbow (right) angles recorded at the BMM (top) and at the WAM (bottom). Red lines: angles generated by the BMM and sent to the WAM PC; blue dotted lines: actual angles where the WAM arm moved and which were fed back to the BMM; green lines: target angles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Please cite this article in press as: Dura-Bernal, S., et al. Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm. Pattern Recognition Lett. (2013), http://dx.doi.org/10.1016/j.patrec.2013.05.019

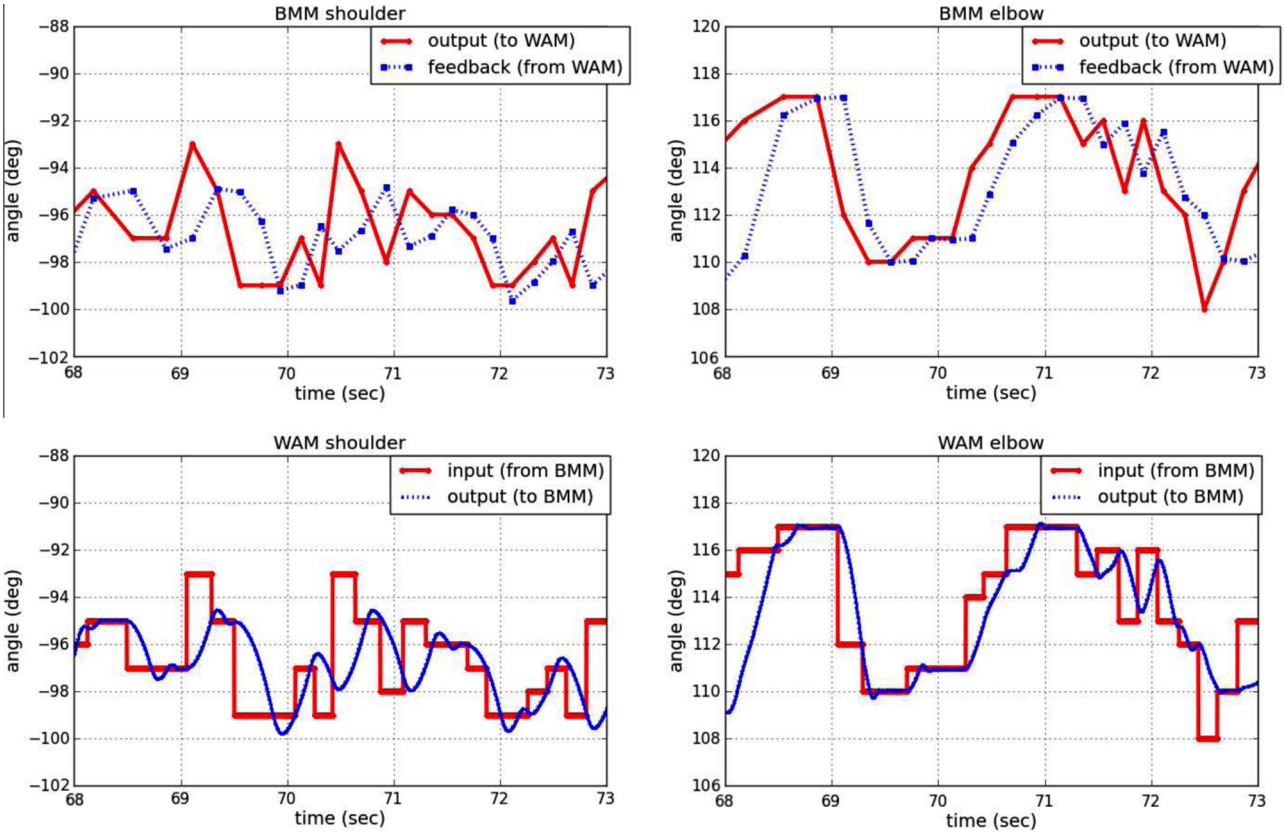

Fig. 7. Comparison of joint angles (5 s zoom). Same as Fig. 6 but zoomed in 20× to illustrate discrepancy in update frequencies (~6 Hz for BMM and 500 Hz for WAM) and how the WAM smoothes the BMM instantaneous trajectory changes.

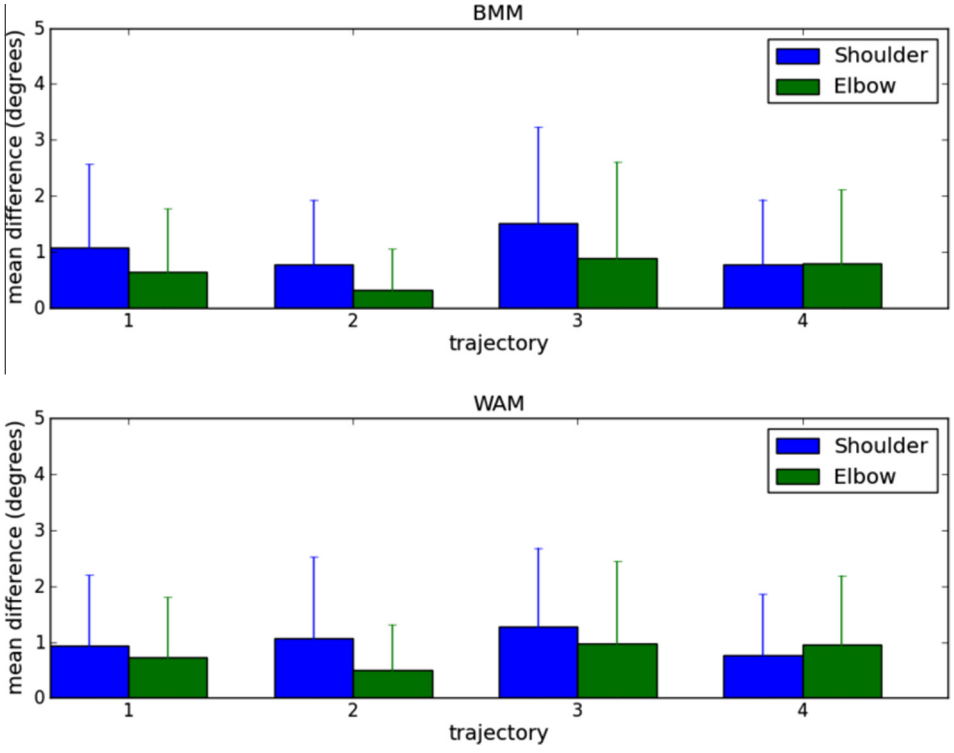

Fig. 8. Difference between angles. Mean and standard deviation (error bars) of the difference between the angles sent to and received from the WAM, recorded at the BMM (top); and received from and sent to the BMM, recorded at the WAM (bottom).

Please cite this article in press as: Dura-Bernal, S., et al. Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm. Pattern Recognition Lett. (2013), http://dx.doi.org/10.1016/j.patrec.2013.05.019

neural populations, sending and receiving packets to/from the WAM arm, and sending packets to V-REP for visualization. The time per iteration was similar accross trajectories (right panel). These results indicate that the overall temporal bottleneck was not imposed by the network interfaces, but by the BMM itself, which only sent and received packets at every 50 ms (simulated-time) iteration, with real-time per iteration varying depending on the amount of processing required.

3.3. Shoulder and elbow angles

We obtained good correspondence between BMM target angles and resultant WAM angles (Fig. 6). Angles sent by the BMM (red) oscillate near the target lines (green), reflecting ongoing babble in the system. WAM (blue) followed with excellent accuracy over the full 95 clock-sec simulation.

The detailed angle trajectories shown in Fig. 7 demonstrated delay of 100–200 ms, as well as some undershoot. Undershoot was greater at the shoulder than at the elbow due to the greater inertia at the proximal joint. Markers provide precise times when packets were sent or received from/to BMM (top), illustrating the time per update iteration, summarized in Fig. 5 (right panel). Angles recorded at the WAM, with sampling at 2 ms, look continuous at this time scale (Fig. 7 bottom). The horizontal red line segments show the intermittency of packets received from the BMM, demonstrating persistence of angles from the prior packet while awaiting the next packet's arrival. The abrupt (instantaneous) angle change commands from the BMM must then be smoothed for the WAM by imposing the speed constraints using the techniques described in *Methods*.

To quantify the accuracy of the trajectory followed by the WAM, we compared angles sent to and received from the WAM, recorded at the BMM; and those sent to and received from the BMM, recorded at the WAM (Fig. 8). We discarded angle differences due to time delay by comparing the sent trajectory at iteration n with the received trajectory at iteration $n+1$. Mean angle difference recorded at the BMM, averaged for the two joints and all trajectories was 0.84 degrees. Taking into account the range of operation of the WAM joints is 226 degrees (shoulder) and 230 degrees (elbow), the overall relative angle difference amounts to only 0.36%.

3.4. Spatiotemporal trajectory

As noted in our previous papers (Chadderdon et al., 2012), the presence of babble produces irregular spatiotemporal paths from starting point to target (Fig. 9). The four trajectories tested were however closely matched from the BMM commands (red) to the WAM end-effector with feedback to the BMM (blue).

The location difference recorded at the BMM, averaged over all trajectories, was 1.29 cm. Given that the range of operation of the WAM arm is 2.02 m, this indicates a relative end-effector location difference of only 0.64%.

4. Discussion

We have developed a real-time interface between a BMM of sensorimotor cortex and a robotic device in the real-world. We used our model to demonstrate the feasibility of using realistic neuronal network models to control devices in real-time. We evaluated the system using four different reaches each lasting greater than 1 min. We demonstrated that the robot arm could follow these BMM trajectories in real time.

The trajectories generated by the BMM exhibited significant fluctuations as a result of the babble noise used to produce random movement for modulation through reinforcement learning. An additional factor was the large discrepancy between the update frequency of the model (~6 Hz) compared to that of the robot arm (500 Hz). These two factors made it physically impossible

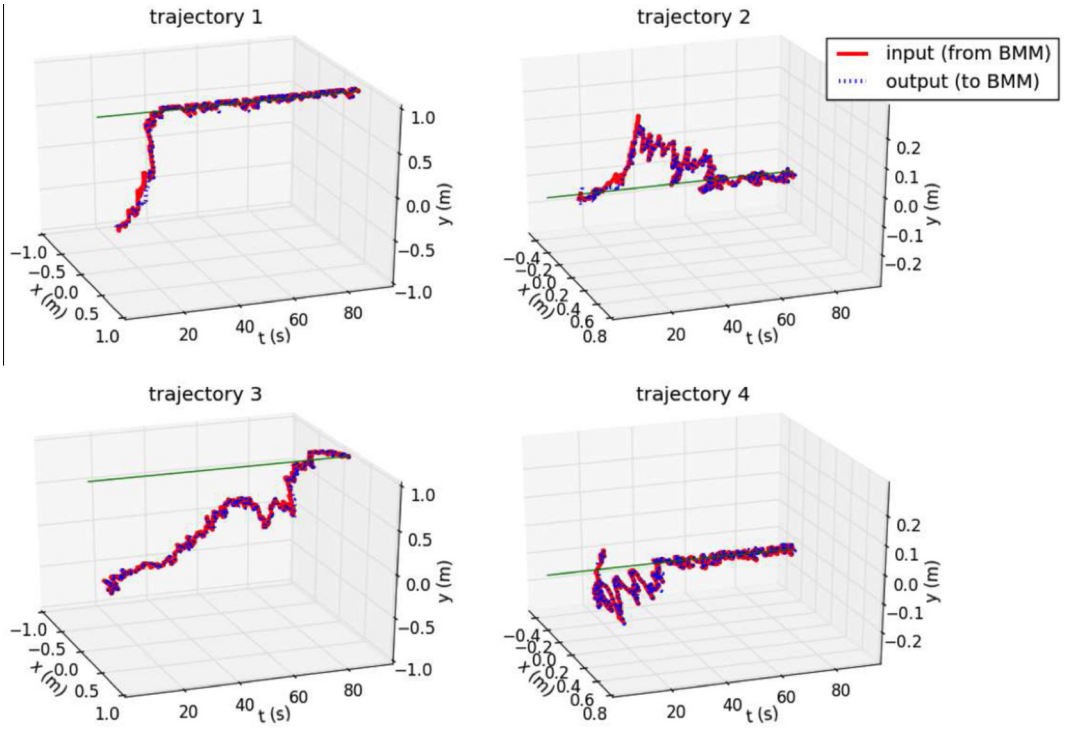

Fig. 9. Spatiotemporal trajectories of arm end-effector. The horizontal and vertical locations over time of the WAM end-effector, sent to (red) and received from (blue) the WAM, based on the recorded angles and the robot dimensions. Green line represents the target. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Please cite this article in press as: Dura-Bernal, S., et al. Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm. Pattern Recognition Lett. (2013), http://dx.doi.org/10.1016/j.patrec.2013.05.019

for the arm to follow the abrupt instantaneous changes of several degrees received from the model. However, in this study we are not evaluating the BMM itself but rather its interface with a robotic arm. In this context, the large fluctuations and the discrepancy in update frequency provided a greater challenge to producing a workable system, which we were able to solve by imposing a fixed speed at each joint. In this way, the arm's control system was able to generate an approximate smoothed version of the trajectory for the arm to follow. This explains the small differences in the trajectories sent to and received from the WAM.

In order to improve the trajectories generated by our BMM we have begun work on adding an intermediate step between the BMM and the WAM robotic arm: a musculoskeletal arm model. The interposition of this arm should increase the biological realism of our motor control system. A virtual musculoskeletal arm model will take as input neural excitation for each muscle and will provide realistic limb position information including muscle fiber length, tendon length, and force or joint angles. The model will then feed muscle neural activation patterns to the virtual arm, whose output will then be used to control the robot arm, leading to more realistic movements. This feedback information, whether from the robotic arm or from the musculoskeletal arm model, will then provide the position information used for reinforcement learning in the brain model. Additionally, we are parallelizing our BMM so that it can run in our high performance computing (HPC) cluster leading to a higher update rate and smoother trajectories.

The software/hardware design reported here sets the stage for our future work, which will include a closed-loop brain–machine interface with a non-human primate. This interface will acquire signals from the primate's brain, pass it to the BMM, and pass motor commands to the robotic arm. The monkey will receive visual and sensory feedback and then modulate its brain state, affecting the signals it sends to the sensorimotor cortex model. This type of system will be a new form of brain–machine interface where the robotic and biological sides are both learning to work together (Digiovanna et al., 2010; Sanchez et al., 2012).

As our models become more realistic, we will use them as stand-ins for actual brain regions that are damaged or temporarily inactivated. For example, a biomimetic brain model might take input from dorsal premotor cortex (PMd) in subjects that have damage to the motor cortex (M1) and serve to translate the PMd command signals into reaching movements that would have been generated in undamaged M1. These models will also be used to predict the results of *in vivo* experiments. We have recently developed computer models of sensory cortex, and modeled the effects of electrical microstimulation (Kerr et al., 2012). At a later stage, it should be possible to use signals from the virtual or robotic limb to generate microstimulation signals for sensory areas of brain to allow a user to feel his or her prosthetic limb.

Acknowledgements

Research supported by: DARPA grant N66001-10-C-2008. The authors thank Barrett Technology Support Team for WAM arm support; and Larry Eberle (SUNY Downstate) for Neurosim lab support.

References

Almassy, N., Edelman, G., Sporns, O., 1998. Behavioral constraints in the development of neuronal properties: a cortical model embedded in a real-world device. Cereb Cortex 8 (4), 346–361.

Barrett Technology Inc., 2012a. WAM User Manual.

Barrett Technology Inc., 2012b. Libbarrett C++ library. Available at: <http://web.barrett.com/libbarrett>.

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J.M., Diesmann, M., Morrison, A., Goodman, P.H., Harris, F.C.J., Zirpe, M., Natschlager, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A.P., El Boustani, S., Destexhe, A., 2007. Simulation of networks of spiking neurons: a review of tools and strategies., J. Comput. Neurosci. 23 (3), 349–398, http://dx.doi.org/10.1007/s10827-007-0038-6. ISSN 0929-5313 (Print); 0929-5313 (Linking).

Buzsáki, G., 2004. Large-scale recording of neuronal ensembles. Nature neuroscience 7 (5), 446–451.

Carmena, J.M., Lebedev, M.A., Crist, R.E., O'Doherty, J.E., Santucci, D.M., Dimitrov, D.F., Patil, P.G., Henriquez, C.S., Nicolelis, M.A.L., 2003. Learning to Control a Brain–Machine Interface for Reaching and Grasping by Primates. PLoS Biol. 1 (2), e42. http://dx.doi.org/10.1371/journal.pbio.0000042.

Carnevale, N., Hines, M., 2006. The NEURON Book. Cambridge University Press, New York.

Chadderdon, G., 2009. A neurocomputational model of the functional role of dopamine in stimulus-response task learning and performance, Ph.D. thesis. Indiana University. Available at: <http://pqdtopen.proquest.com/#viewpdf?dispub=3355003>.

Chadderdon, G.L., Neymotin, S.A., Kerr, C.C., Lytton, W.W., 2012. Reinforcement learning of targeted movement in a spiking neuronal model of motor cortex. PLoS One 7 (10), e47251. http://dx.doi.org/10.1371/journal.pone.0047251.

Digiovanna, J., Rattanatamrong, P., Zhao, M., Mahmoudi, B., Hermer, L., Figueiredo, R., Principe, J.C., Fortes, J., Sanchez, J.C., 2010. Cyber-workstation for computational neuroscience. Front. Neuroeng. 2, 7–19, http://dx.doi.org/10.3389/neuro.16.017.2009. ISSN 1662-6443 (Electronic); 1662-6443 (Linking).

Edelman, G., 1987. Neural Darwinism: The Theory of Neuronal Group Selection. Basic Books New York, New York.

Edelman, G., 2006. The embodiment of mind. Daedalus 135 (3), 23–32.

Freese, M., Singh, S., Ozaki, F., Matsuhira, N., 2010. Virtual Robot Experimentation Platform V-REP: A Versatile 3D Robot Simulator. In: N. Ando, S. Balakirsky, T. Hemker, M. Reggiani, O. Stryk (Eds.), Simulation, Modeling, and Programming for Autonomous Robots. Lecture Notes in Computer Science, vol. 6472. Springer, Berlin, Heidelberg. http://dx.doi.org/10.1007/978-3-642-17319-68. ISBN 978-3-642-17318-9, 51-62.

Izhikevich, E., 2007. Solving the distal reward problem through linkage of STDP and dopamine signaling 17, 2443–2452.

Joel, D., Niv, Y., Ruppin, E., 2002. Actor-critic models of the basal ganglia: new anatomical and computational perspectives. Neural Networks 15 (4–6), 535–547.

Kerr, C., Neymotin, S., Chadderdon, G., Fietkiewicz, C., Francis, J., Lytton, W., 2012. Electrostimulation as a prosthesis for repair of information flow in a computer model of neocortex. IEEE Trans. Neural Syst. Rehabil. Eng. 20 (2), 153–160.

Krichmar, J., Edelman, G., 2005. Brain-based devices for the study of nervous systems and the development of intelligent machines. Artif Life 11 (1–2), 63–77.

Lungarella, M., Sporns, O., 2006. Mapping information flow in sensorimotor networks. PLoS Comput. Biol. 2 (10), e144.

Lytton, W.W., 2012. Computational neuroscience. In: Encyclopedia of the Neurological Science. Elsevier.

Lytton, W., Omurtag, A., 2007. Tonic-clonic transitions in computer simulation. J Clin Neurophys 24, 175–181.

Lytton, W., Stewart, M., 2005. A rule-based firing model for neural networks. Int J Bioelectromagnetism 7, 47–50.

Lytton, W., Stewart, M., 2006. Rule-based firing for network simulations. Neurocomputing 69 (10–12), 1160–1164.

Lytton, W., Omurtag, A., Neymotin, S., Hines, M., 2008. Just-in-time connectivity for large spiking networks. Neural Comput 20 (11), 2745–2756.

Lytton, W., Neymotin, S., Hines, M., 2008. The virtual slice setup. J Neurosci Methods 171, 309–315.

Mahmoudi, B., Sanchez, J., 2011. A symbiotic brain–machine interface through value-based decision making. PloS One 6 (3), e14760.

MathWorks, Inc., 2012. MATLAB Release 2012b.

Neymotin, S.A., Lee, H., Park, E., Fenton, A.A., Lytton, W.W., 2011. Emergence of physiological oscillation frequencies in a computer model of neocortex. Front. Comput. Neurosci. 5, 19, http://dx.doi.org/10.3389/fncom.2011.00019. ISSN 1662-5188 (Electronic); 1662-5188 (Linking).

Neymotin, S., Lee, H., Park, E., Fenton, A., Lytton, W., 2011b. Emergence of physiological oscillation frequencies in a computer model of neocortex. Front. Comput. Neurosci. 5, 19, http://dx.doi.org/10.3389/fncom.2011.00019.

Neymotin, S.A., Chadderdon, G.L., Kerr, C.C., Francis, J.T., Lytton, W.W. Reinforcement learning of 2-joint virtual arm reaching in a computer model of sensorimotor cortex. Neural Computation, submitted for publication.

Peterson, B., Healy, M., Nadkarni, P., Miller, P., Shepherd, G., 1996. ModelDB: an environment for running and storing computational models and their results applied to neuroscience. J. Am. Med. Inform. Assoc. 3 (6), 389–398.

Sanchez, J., Lytton, W., Carmena, J., Principe, J., Fortes, J., Barbour, R., Francis, J., 2012. Dynamically repairing and replacing neural networks: using hybrid computational and biological tools. IEEE Pulse 3 (1), 57–59, http://dx.doi.org/10.1109/MPUL.2011.2175640. ISSN 2154-2287 (Print).

Sutton, R., Barto, A. 1998. Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, USA.

Thorndike, E., 1911. Animal Intelligence. Macmillan, New York.

Webb, B., 2000. What does robotics offer animal behaviour? Anim. Behav. 60 (5), 545–558.

Please cite this article in press as: Dura-Bernal, S., et al. Towards a real-time interface between a biomimetic model of sensorimotor cortex and a robotic arm. Pattern Recognition Lett. (2013), http://dx.doi.org/10.1016/j.patrec.2013.05.019