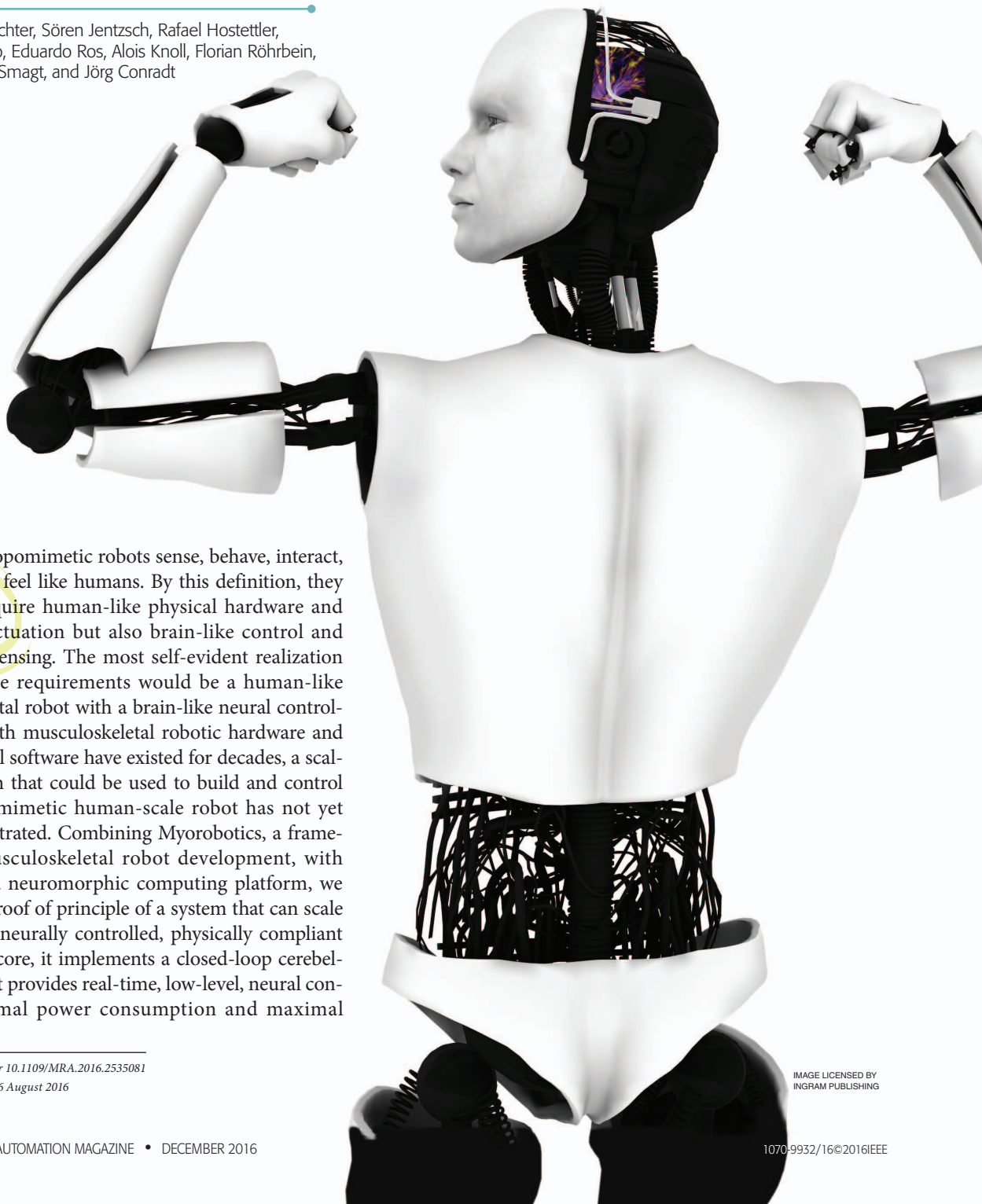
# Musculoskeletal Robots

## *Scalability in Neural Control*

By Christoph Richter, Sören Jentzsch, Rafael Hostettler, Jesús A. Garrido, Eduardo Ros, Alois Knoll, Florian Röhrbein, Patrick van der Smagt, and Jörg Conradt



Anthropomimetic robots sense, behave, interact, and feel like humans. By this definition, they require human-like physical hardware and actuation but also brain-like control and sensing. The most self-evident realization to meet those requirements would be a human-like musculoskeletal robot with a brain-like neural controller. While both musculoskeletal robotic hardware and neural control software have existed for decades, a scalable approach that could be used to build and control an anthropomimetic human-scale robot has not yet been demonstrated. Combining Myorobotics, a framework for musculoskeletal robot development, with SpiNNaker, a neuromorphic computing platform, we present the proof of principle of a system that can scale to dozens of neurally controlled, physically compliant joints. At its core, it implements a closed-loop cerebellar model that provides real-time, low-level, neural control at minimal power consumption and maximal

*Digital Object Identifier 10.1109/MRA.2016.2535081*
*Date of publication: 26 August 2016*





1070-9932/16©2016IEEE

extensibility. Higher-order (e.g., cortical) neural networks and neuromorphic sensors like silicon retinae or cochleae can be incorporated.

## Combining Hardware and Computer Architecture

A major challenge and vision for articulated robots is to behave and interact with humans in a safe and natural manner. Robots that mimic the mechanical properties of the human build strive toward both attributes simultaneously [1], [2], as, by design, they possess built-in compliance and relatively natural—i.e., human-like—mass distribution and dynamics. Musculoskeletal robots in particular offer lightweight, low-inertia end effectors because the main actuators, the skeletal muscles, can be kept at rest. Figure 1 shows a design that coarsely mirrors a human arm. Most of the muscle mass is rigidly attached to the torso. Muscles connect to the distal bone only via tendons, which have a negligible weight. In this way, two passive safety aspects, which minimize the head injury criterion [1], are intrinsic to the anthropomimetic musculoskeletal architecture: compliance and minimal moving mass.

Similarly bioinspired approaches on the controller side are simulated or emulated biological neural networks, because the human brain and central nervous system are the most relevant reference for natural control of musculoskeletal limbs. Neural control as done by animals or humans is the most elegant, versatile, and energy-efficient way to use musculoskeletal systems. Just as the human-like mechanical build has inherent passive safety advantages, brain-like control has desirable active safety features. The human nervous system implements active compliance on multiple levels. Arguably more importantly, though, humans are perfectly accustomed to human-like behavior. Despite the fact that your colleagues could, if so inclined, injure you or others, working with humans is generally considered safe and does not require any special training. Consequently, there is every hope that their natural, and, in this sense, predictable behavior could gain anthropomimetic robots human-like safety attributes. The most demanding requirements and challenges on both the robotic hardware and the controller side are scalability and usability. Anthropomimetic robots have been built by numerous research groups, such as the Jouhou System Kougaku Laboratory of the University of Tokyo and partners within the European Union-funded project Embodied Cognition in a Compliantly Engineered Robot (Eccerobot) [3], [4], among others. However, those systems were custom designed, mostly using complex hardware and software that inhibits reproduction across labs and involves high production costs [5]. The situation is similar with computing platforms. Robotic applications require flexible interfaces and strict real-time execution of large neural simulations [6]. Different neuromorphic architectures and neuroaccelerators have been developed, yet most of them, like those based on graphics processing units [7], [8], lack in terms of scalability. Special-purpose systems like those based on field-programmable gate arrays (FPGAs) [9], [10] or custom silicon [11], [12] are usually too inflexible for a nonexpert to implement and investigate custom learning rules, synapse types, or cell models.

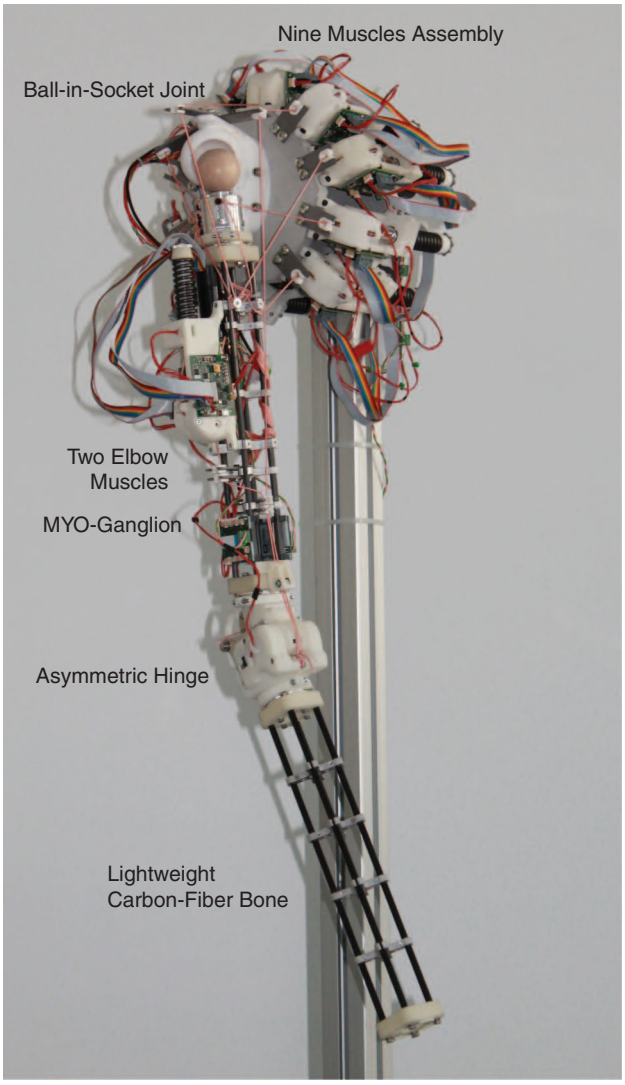



**Figure 1.** The complex Myorobotics arm mimicking the complexity of a human arm without spatula. Nine muscles cooperate to control the ball-in-socket joint. One of these muscles, relating to the biceps, is biarticular, as it is attached so that it affects the motion of two joints, effectively coupling the shoulder and elbow joint.

**Musculoskeletal robots in particular offer lightweight, low-inertia end effectors because the main actuators, the skeletal muscles, can be kept at rest.**

To this end, the prevailing architecture for neural simulations and neural controllers is still the desktop computer, which we define in the context of this work as a Von Neumann architecture with a modest number of computing cores that share a common large random access memory (RAM). Depending on the underlying

computations, such architectures are typically not optimal for simulating large neuronal networks (the human cerebellum alone comprises more than $10^{11}$ neurons [13]), which are inherently parallel [12].

**Whereas the maximum system size might not be relevant to the robotics community, the scalability, power efficiency, flexibility, and ease of use certainly are.**

In this article, we present the unique combination of musculoskeletal robotics hardware (Myorobotics) and neural control substrate implemented on a scalable spiking neural network (SNN) infrastructure (SpiNNaker). We demonstrate how these technologies can address the aforementioned challenges and facilitate the development of human-scale anthropomimetic systems that are controlled by brain-like SNNs. It is our conviction that SpiNNaker and Myorobotics pave the way for large-scale, complex neurorobots.

### SpiNNaker

SpiNNaker [14] is a computer system designed for real-time simulations of SNNs by the Manchester Advanced Processor Technologies Research Group. A typical SpiNNaker system comprises thousands of ARM968 processing cores, which can run arbitrary code. They are distributed on a quasi-seamlessly extensible mesh network that is spanned by special multicast routers at its nodes. SpiNNaker's multicast routers are optimized for small (40 or 72 b wide) data packets. Those SpiNNaker packets typically resemble action potentials or neural spikes in an SNN simulation. They typically convey only the source address of their originating neuron, from which the routers deduce the routing direction based on a user-programmed routing table. Every SpiNNaker chip houses one router, 18 SpiNNaker cores (each with 96 kB of local memory), and 128 MB of shared synchronous dynamic RAM.

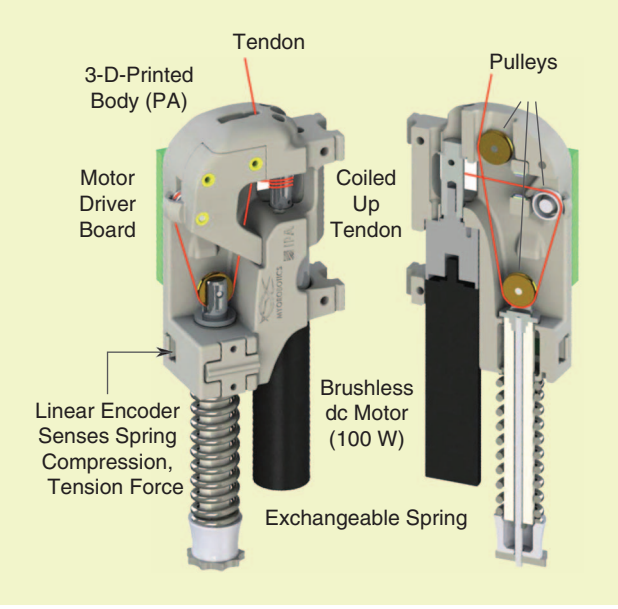



**Figure 2.** The Myorobotics muscle with its components. The tendon (red cable) is routed in a triangular fashion in the muscle to create a nonlinear net spring force. The tendon force is sensed by measuring the spring displacement through a magnetic strip fixed to the guiding rod of the spring that slides by a hall-effect encoder. This allows calculation of the respective force from a known spring constant and tendon routing geometry.

The Manchester group provides an open software framework that promotes an event-driven programming model through the Spin1 application programming interface [14]. Implementations of PyNN, a common interface for neuronal network simulators [15], and Nengo, a graphical and scripting-based software package for simulating large-scale neural systems [16], are provided as a high-level, user-friendly way to specify neural networks. These networks are then automatically mapped, uploaded, and executed on SpiNNaker. The entire software framework is open source, so it can be extended and modified by its users (see https://github.com/SpiNNakerManchester).

In terms of SNN simulation performance, SpiNNaker is superior to desktop computers by orders of magnitude. As a rule of thumb, a single SpiNNaker chip ($P \approx 1$ W) can handle a network of 10,000 leaky integrate-and-fire neurons in real time. A desktop computer needs a high-performance processor ($P \approx 50$ W) with fast memory to perform the same task. A single SpiNN-5 board contains 48 SpiNNaker chips drawing about the same amount of electrical power ($P \approx 50$ W) but providing about 50 times the computational power. Finally, the system scales from 18 (single chip) to over 1 million cores (57,600 chips), so the system size can be adapted to a wide range of neural network sizes by interconnecting an appropriate number of SpiNNaker boards. Whereas the maximum system size might not be relevant to the robotics community, the scalability, power efficiency, flexibility, and ease of use certainly are. SpiNNaker is designed for real-time SNN simulations. Given proper interfaces, it offers the prime opportunity to let large SNNs interact with and adapt to the real world.

### Myorobotics

Myorobotics is a tool kit for modular musculoskeletal robots that encompasses the full life cycle of robot design. Robots can be assembled, optimized, and simulated from primitives, then built and controlled from the same software. The robots are assembled from a set of primitives: bones, muscles, and joints, which are shown in Figure 1. The most interesting of those building blocks, the muscle, is detailed in Figure 2. Its body is made of three-dimensional (3-D) printed polyamide (PA). It is actuated by a 100 W dc motor (Maxon Motor EC series) that coils up a cable—the tendon. Three pulleys route the tendon in a triangular fashion. One of the pulleys is attached to a

spring-loaded guiding rod. This mechanism endows the Myorobotics actuator with a (nonlinear) series elasticity.

The Myorobotics tool kit allows for the creation of a multitude of robot morphologies and enables researchers to investigate properties and dynamics of musculoskeletal robots. Its dedicated electronics provide tendon force, velocity, position, and torque control at 500 Hz directly from a standard desktop computer with all sensory data available on the bus. At this update rate, the bandwidth of a single FlexRay interface, which is employed for high-level control, allows for up to 24 motors that can be driven concurrently.

The framework can be extended easily with new primitives thanks to a standardized structure connector between the parts as well as a software plug-in that imports the construction directly from the computer-aided design software SolidWorks. Consequentially, the system allows for primitives from a broad range of categories and covers many interesting use cases, such as anthropomimetic arms with complex shoulder joints (Figure 1), quadrupeds, and hopping robots. As the whole system was built with the nonrobotic expert user in mind, it is easy to use, and it allows for fast modification of the robot topology. The entire system, including all three-dimensional (3-D) models, schematics, and all source code, is open source (see http://www.myorobotics.eu/).

What differentiates Myorobotics from other series elastic actuators and variable stiffness actuators is that Myorobotics actuators generate pulling forces between two attachment points rather than torques between two rigid links. This yields a fundamentally different control problem. While it can be reduced to classical joint angle-based control by describing a muscle Jacobian that maps the lengths of all tendons that apply forces between two links to a joint angle, this mapping is, in many cases, not unique; choosing a specific mapping means reducing the space of possible trajectories. However, there is ongoing research to design control strategies that directly map task space trajectories to desired muscle forces without the intermediate step of calculating target joint angles. This is especially interesting in the context of this article, as all biological muscle-based systems solve this control problem rather than a target joint angle/torque problem. Myorobotics is thus a much closer model of the behavior of biological musculoskeletal systems than series elastic actuators such as the mechanically adjustable compliance and controllable equilibrium position actuator.

### *Neural Circuitry*

We focus on implementing a cerebellar model to control the dynamics of the robotic system. Even though the major role of the cerebellum seems to be supervised learning of motor patterns [17], it is clear that vertebrate limb control cannot be reduced to cerebellar functioning [18]. An individual with cerebellar lesions may be able to move the arm to successfully reach a target and to successfully adjust the hand to the size of an object. However, the action cannot be made swiftly and accurately, and the ability to coordinate the timing of the two subactions is lacking [19]. Vertebrate movement generation involves the basal ganglia, filtering out unwanted movements [17], as well as the motor and parietal cortices. Movement realization, of course, also involves the spinal cord, which controls antagonism and seems to take care of nonlinearities in muscular functionality. Our model, however, focuses on a model of the cerebellar neurocircuitry for the following reasons. First, the fast learning of cerebellar circuitry is important for fast adaptation to environmental influences [20]. Second, some functionality of the spinal cord can be simulated with simple proportional-integral differential (PID) controllers [21], especially for the comparatively simple actuator behavior that our system exhibits.

**As the whole system was built with the nonrobotic expert user in mind, it is easy to use, and allows for fast modification of the robot topology.**

## Setup

### *Robot*

The robot employed in our proof of concept is the most basic setup that can be built with Myorobotics, consisting of a single symmetric hinge joint with two bones and two muscles driving it (Figure 3). The system uses only the motor driver boards from the Myorobotics electronics, which we interface using a controller area network (CAN) bus. Larger Myorobotics systems connect the driver boards to intermediate controller boards (MYO-Ganglia) that offer a higher-level, higher-bandwidth control interface via FlexRay.

Figure 3(a) highlights the individual parts of our joint assembly. The two artificial muscles ($m_1$, $m_2$) are connected to the lower bone ($b_l$). Tendons connect them to the opposite

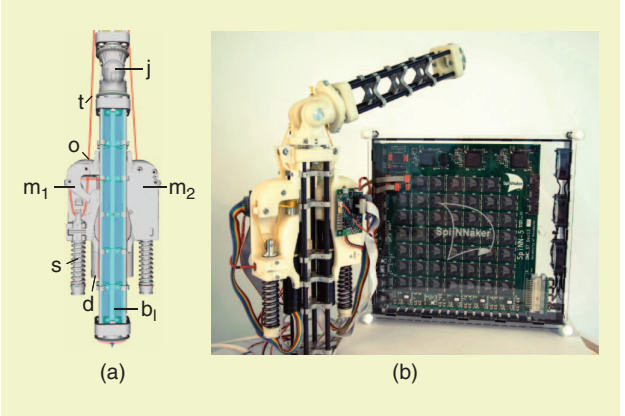



**Figure 3.** The single-joint Myorobotics proof-of-principle setup shown as (a) a schematic and (b) a photograph with a SpiNN-5 48-chip SpiNNaker neuromorphic computer. $m_1$ and $m_2$: two artificial muscles; $b_l$: the lower bone; j: the hinge joint; d: the brushless dc motor; t: the tendon; s: the spring; and o: the fixed outlet.

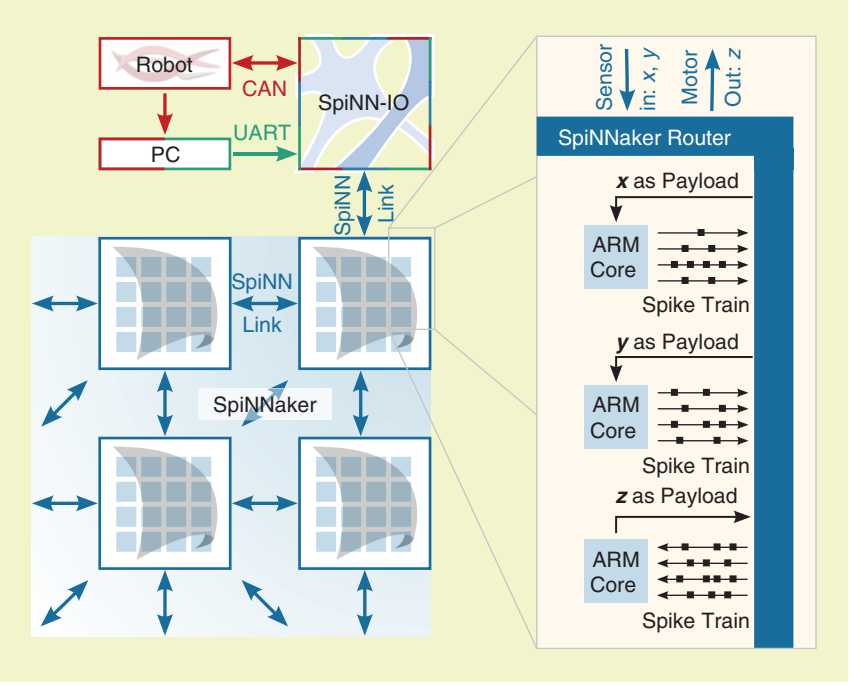



**Figure 4.** The logical and electrical layout of our system. The frame and arrow colors indicate bus types. The SpiNN-IO board provides a real-time interface between the robot, the desktop computer, and SpiNNaker. Communicating with a SpiNNaker chip's router via SpiNN-Link provides the input SpiNNaker cores with sensory data *x*,*y*. It receives motor commands *z* from output cores that it translates and forwards to the robot.

side of the hinge joint (j). Each muscle consists of a brushless dc motor (d) that coils up the tendon (t); we will call this the *actuator*. The tendon is routed via a spring (s) and exits at a fixed outlet (o). The mechanically linear spring is combined with a triangular routing of the tendon (Figure 2), making the net spring behavior nonlinear. Because the actuators can only pull, an antagonist actuator is required. By pretensioning both actuators, both springs get contracted, thereby changing the mechanical stiffness of the system.

**All settings like time constants or scale factors can be adjusted in a user-friendly, object-oriented fashion.**

### Interfaces

To connect SpiNNaker to robotic sensors and actuators, we have developed a hardware interface that acts like another node on SpiNNaker's mesh network [22]. It translates sensor data into SpiNNaker packets and SpiNNaker packets into, for example, motor commands. The microcontroller-based design allows us to connect SpiNNaker to many different bus systems, including universal asynchronous receiver/transmitters (UARTs) and CANs. We use the former for communication with an external desktop computer, the latter for Myorobotics actuators and sensors (Figure 4). Although the interface board allows for real-time injection of neural spike trains into SpiNNaker, our current implementation saves bandwidth by handling the de- and encoding between robot data and neural spikes directly on SpiNNaker. The inset in Figure 4 illustrates this setup: sensory updates arrive as a SpiNNaker packet's payload at the respective ARM cores that are continuously emitting spike trains encoding the current sensory state. Likewise, dedicated motor cores continuously translate incoming spike trains to motor commands.

The typical translations performed on the input SpiNNaker cores are either rate or population coding, the latter with Gaussian receptive fields and linearly distributed preferred values. Because SpiNNaker cores can be freely programmed, more flexible translation schemes, for example, involving self-organizing maps, can be implemented. Our output cores translate the rate of incoming spikes from within the SpiNNaker mesh to a motor output signal via a linear transformation and an exponential falloff in time. The time window and update cycle is typically 20 ms. Again, more complex translation schemes can be implemented. Those could involve proprioceptive feedback from the Myorobotics actuators and, in this way, emulate the macroscopic or microscopic behavior of real skeletal muscles. From a PyNN network point of view, input and output are handled and set up like normal neural populations. The low-level implementation as C code is wrapped by PyNN objects and thus hidden from the PyNN programmer. All settings like time constants or scale factors can be adjusted in a user-friendly, object-oriented fashion.

### Network Model

As a first demonstration of our system, we chose a cerebellar model that has previously been used to operate robots [20]. Our network model is akin to a Marr–Albus style cerebellum [23], [24]. Its specific setup including all cell parameters is derived from [20]. Although several network configurations were evaluated in [20], we have considered the network that receives an implicit estimation of the robot actual state $\phi_{\text{act}}$ and the set point $\phi_{\text{set}}$. Figure 5 illustrates the network structure. The network is composed of leaky integrate-and-fire neurons with biologically realistic cell parameters and plausible divergence/convergence ratios between the different layers. As previously done in [20], we are omitting inhibitory interneurons and the olivo-cerebellar loop to arrive at a most basic and deterministic model. However, the network still

keeps the main roles that have been proposed for each layer in the Marr–Albus model [23], [24], i.e., input sparse recoding of the mossy fiber (MoF) inputs in the granular layer and supervised learning in the Purkinje cells (PuCs).

Each of the two motors is controlled by the spike rate of four deep cerebellar nuclei (DCN) cells, which receive excitatory input from 32 MoFs and inhibitory input from eight PuCs. MoF spiking activity (representing sensory input, actual state and control data, and target angle) produces sequences of active granule cells (GrCs). Because each of the 256 GrCs receives input from a unique set of MoF cells, a sparse coding of the input is made available in the parallel fibers (PFs), the long axons of the GrCs.

The inhibitory corrective term that the DCN receives from PuCs is shaped through supervised learning between PFs and PuCs. The teaching signal encoding the actual error reaches the PuCs through the inferior olive (InO), producing complex spikes. This particular type of long-lasting spikes has been demonstrated to induce long-term depression in the PF-PuC synapses when correlated with simultaneous PF spikes [25]. This learning mechanism has been implemented by using a kernel function $\Delta w(t_{\mathrm{GrC}} - t_{\mathrm{InO}})$ relating mutual InO–GrC spike timing with synaptic weight changes $\Delta w$ (see [20] for details). It basically punishes synapses that likely lead to erroneous behavior: if a GrC spike on a GrC-PuC synapse leads to some action and is followed by an InO spike after a characteristic response time, say 100 ms, then the respective synaptic weight, which was likely responsible for that error, is depressed [26]. To compensate the long-term depression term, long-term potentiation is induced every time a presynaptic spike occurs in the PFs. The effective spike timing-dependent plasticity function $\Delta w(t_{\mathrm{InO}} - t_{\mathrm{GrC}})$ is plotted in Figure 5. Interestingly, this learning rule also deals with the long delay that has been observed in the action-perception loop of the nervous system that has been estimated at around 100 ms [26].

This rather unusual learning rule would be impossible to implement on many neuroaccelerator platforms. SpiNNaker, on the other hand, is freely programmable. Just like the previously discussed input and output populations, we implemented this learning rule as low-level C code on SpiNNaker and wrapped it into a PyNN object for the high-level network description. We chose a lookup-table (LUT)-based approach, in which the LUTs for the temporal kernel are compiled by the Python front end. The respective SpiNNaker cores buffer up to 160 spikes per simulated synapse and evaluate their mutual timing and the corresponding synaptic weight change periodically.

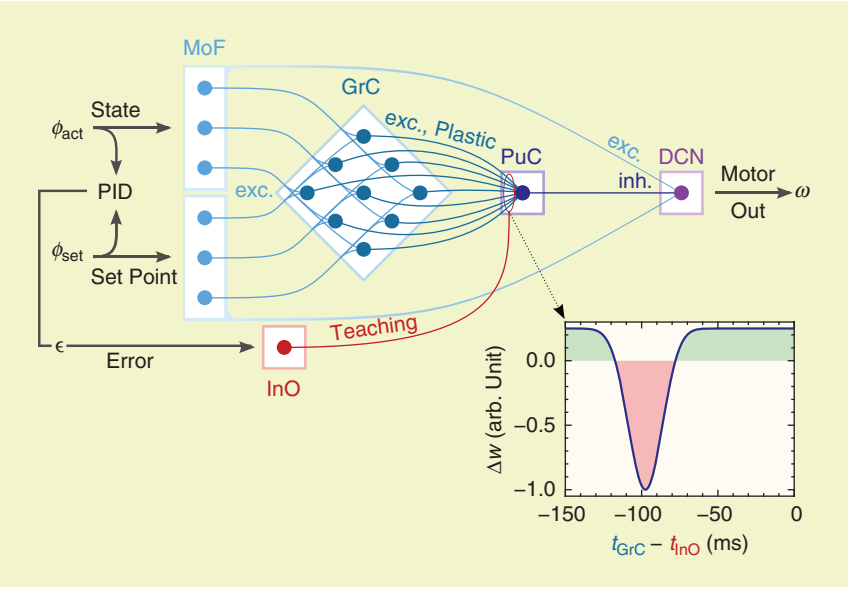



**Figure 5.** A schematic drawing of our cerebellar model along with its input (actual angle $\phi_{act}$, set point $\phi_{set}$, control error $\epsilon$) and output (motor pulsewidth $\omega$). Single neurons or spike sources are represented by dots, populations are marked by rectangular frames, and the thin lines connecting neurons represent synapses. The graph represents the learning rule governing weight changes $\Delta w$ at GrC-PuC synapses in response to the relative timing of GrC and InO spikes as they arrive at PuC dendrites.

The reference network implementation, which we ported to SpiNNaker, is running on an event-driven simulator based on LUTs (EDLUT, see http://edlut.googlecode.com) [27], [7], a high-performance event-driven neural network simulator software. We developed a framework that translates a high-level text-based network description for either EDLUT or PyNN, runs the simulation on the PC or SpiNNaker, respectively, and compares the resulting network output. Through the use of a programmable power supply (Manson HCS-3202) we can monitor SpiNNaker's runtime power consumption and compare it to that of EDLUT running on our desktop computer. This way our SpiNNaker implementation could be rigorously checked and tested against the reference implementation on EDLUT.

Our SpiNNaker implementation matches the EDLUT reference well. Minor deviations mainly stem from the fact that the SpiNNaker implementation is tick based and uses fixed-point representations, while EDLUT is purely event driven and uses double precision floating point. SpiNNaker cores lack a floating point unit, so floating point computations on SpiNNaker would be inefficient. The Manchester team made this design decision to save on power consumption and die area per core. A comparison to the relatively efficient desktop-based software simulator EDLUT highlights SpiNNaker's power efficiency. Depending on the network layout and its input, SpiNNaker's energy consumption is just one hundredth to one tenth that of EDLUT running on a typical desktop computer—a considerable asset especially relevant for autonomous robots.

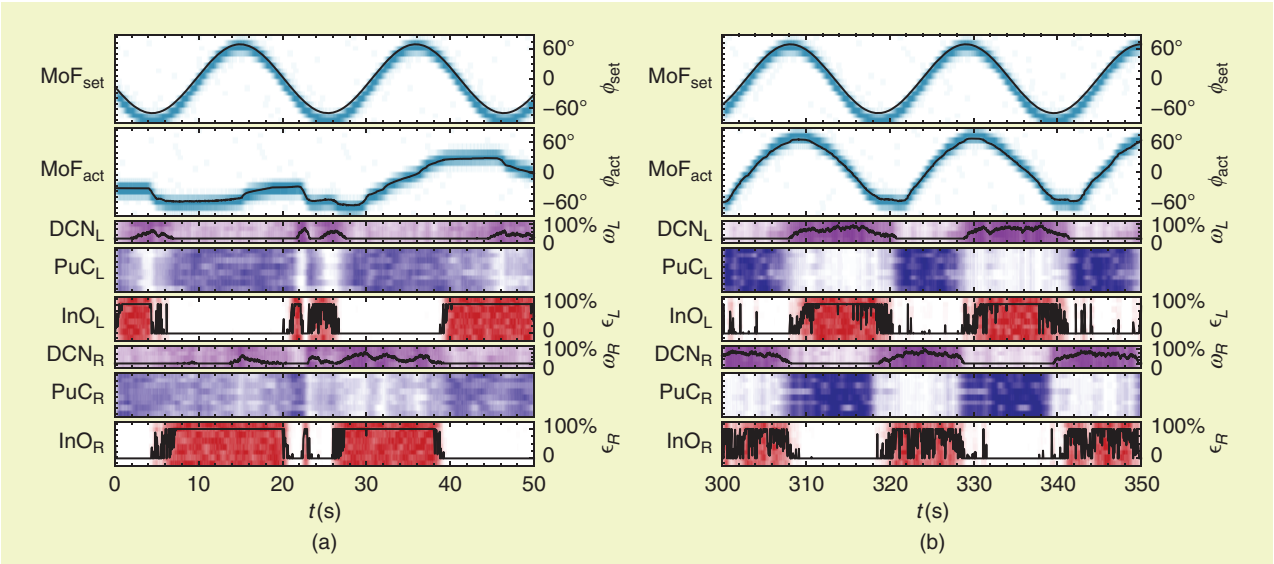



**Figure 6.** The performance of our cerebellar real-time simulation on SpiNNaker. (a) Naive cerebellum and (b) five minutes later: trained cerebellum. Colored semiopaque squares symbolize spike events (spike density $\propto$ color saturation), left ordinate indicates the neuron ID. The black solid curves indicate the respective angle $\phi$ (MoF input), motor output $\omega$ (DCN output), or error signal $\epsilon$ (InO input). Indices: *L* and *R*, left and right; act, actual sensory values; set, set point.

### *Graphical User Interface*

While our system can run headless, in a closed-loop fashion, we have built a graphical user interface (GUI) for live monitoring and interaction with the neural simulation on SpiNNaker. The software runs on an external computing station, receives data from the CAN bus, and uses UART to inject data into SpiNNaker via the SpiNN-IO board (Figure 4). Through the GUI, the user can control the target joint angle of the robot and adjust the PID parameters determining the teaching (error) signal calculation. The teaching signal as well the target angle are sent to SpiNNaker at an update frequency of 20 Hz. As for debugging purposes and performance evaluation, the user can monitor the current CAN data, the deviation between current and target joint angle, and the current error signal as well as the live spike train of selected neuron populations.

**Through the GUI, the user can control the target joint angle of the robot and adjust the PID parameters determining the teaching (error) signal calculation.**

## Evaluation

Figure 6 shows the performance of our proof-of-principle cerebellar model running in real time on SpiNNaker while controlling the antagonistic Myorobotics joint. It illustrates neural spikes as colored raster plots and its control and sensory input values over time as black solid lines, where applicable. There are separate PuC (dark blue spikes), InO (red spikes), and DCN (purple spikes) populations, for the left (index *L*) and right (index *R*) actuator. The error signals $\epsilon_L$, $\epsilon_R$ are computed as a PID error signal $E(\phi_{\text{set}} - \phi_{\text{act}})$ by the GUI. The corresponding spike trains are emitted by the InO populations. They shape the weights between the GrCs (not shown) and the respective PuCs. The control input is the joint angle set point $\phi_{\text{set}}(t)$ as given by the GUI, which corresponds closely to the MoF spikes of the $\text{MoF}_{\text{set}}$ population. The other control input is the actual joint angle $\phi_{\text{act}}(t)$ as measured by a magnetic angle sensor within the Myorobotics joint. $\text{MoF}_{\text{act}}$ is the corresponding neural population. The SpiNN-IO board reads $\phi_{\text{act}}$ directly from the CAN bus, receives $\phi_{\text{set}}$, $\epsilon_L$, $\epsilon_R$ via UART (from a universal serial bus-UART interface), and streams all values into SpiNNaker via SpiNN-Link, where they are translated into spike trains, as illustrated in Figure 4.

### *Control Performance*

In the present setup the teaching signal conveys the mismatch between the actual and the target joint angle. Consequently, to minimize that error and maximize the agreement between $\phi_{\text{act}}$ and $\phi_{\text{set}}$, the network learns to perform antagonistic control. In our scenario, the network learns to follow the given sinusoidal trajectory $\phi_{\text{set}}(t)$ within a few revolutions. The learning is exclusively based on the intrinsic plasticity mechanism at GrC-PuC synapses, as previously explained in the "Network Model" section. Figure 6(a) shows the performance of the naive network with randomly initialized weights at the GrC-PuC synapses. In this state, the cerebellar model does not even know right from left. Therefore, the joint angle $\phi_{\text{act}}$ ($\text{MoF}_{\text{act}}$) does not track the given trajectory $\phi_{\text{set}}(t)$ ($\text{MoF}_{\text{set}}$) at all. As an example, at $t = 15$ s,

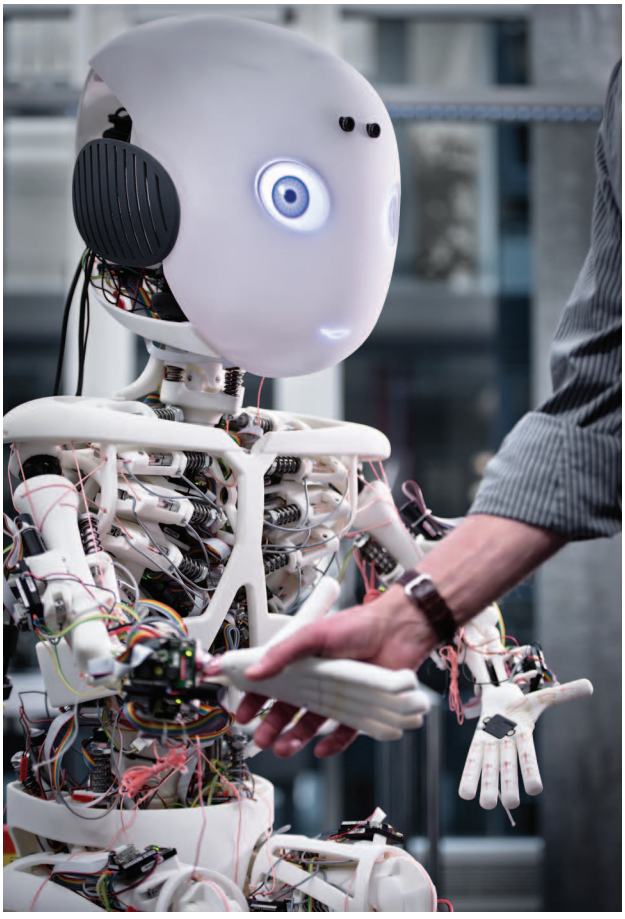


**Figure 7.** Roboy, a human-like, musculoskeletal robot with 28 degrees of freedom and 48 motors, to be controlled by brain-inspired systems. (Photograph courtesy of Adrian Baer.)

$\phi_{\text{set}}$ is at the rightmost position, $\phi_{\text{act}}$ still on the far left side. In this situation, the right muscle should clearly pull more. The teaching signal reacts accordingly: $\epsilon_R$ is at its maximum, resulting in a high $\text{InO}_\text{R}$ firing rate. The corresponding $\text{GrC-PuC}_\text{R}$ weights decrease accordingly. In subsequent similar situations this results in less $\text{DCN}_\text{R}$ inhibition by the $\text{PuC}_\text{R}$ population and more motor output $\omega_R$. So after five minutes of run time (and learning), the system can follow the trajectory much better [Figure 6(b)]: $\phi_{\text{act}}$ tracks $\phi_{\text{set}}$ much more closely, and the cerebellar model has learned to do antagonistic control. The cerebellum can also learn to follow different waveforms or manually controlled trajectories (see https://youtu.be/y6MwOtW3_kQ for a video demonstration).

Note that, in the given example, the network output is the sole control input to the robot arm. It controls the motors directly via pulsewidth modulation. While this nicely demonstrates the learning capabilities of the network, it does not mirror the biological antetype. In a more biologically realistic scenario, the cerebellum would output a corrective term that adds to a (cortical) forward-kinematic control signal.

### *Scalability and Constraints*

Our present configuration runs on a single SpiNNaker chip. It utilizes only 16 SpiNNaker cores, 2% of a single SpiNN-5 board. Consequently, there is ample room for adding more joints and actuators as well as higher-level (e.g., cortical) neural networks. With SpiNNaker being a scalable system, computing resources are clearly no longer the bottleneck.

Our SpiNN-IO board connects the robot and the desktop computer with SpiNNaker. Its microcontroller limits the effective, combined update rate of input and output populations to about 500 kHz [22]. In effect, our current system could handle 500 input/output (I/O) populations at an update rate of 1 kHz. As each I/O population occupies a single SpiNNaker core, those neural populations would fill about 60% of a single SpiNN-5 board. A limit of 500 sensory streams at 1 kHz update rate translates to roughly 100 actuators, or dozens of joints that could be controlled with a single or few SpiNN-5 boards—within an order of magnitude to a human-scale robot. This limit could be alleviated by using 1) more than one SpiNN-IO (on separate SpiNN-5 boards), or 2) a modified SpiNN-IO design that uses SpiNNaker's interboard connectors. FPGAs on the SpiNN-5 board multiplex eight SpiNN-Link ports on each connector.

The remaining bottleneck is the communication between SpiNN-IO and the robot. In our current setting, we can use up to four separate CAN buses, which can manage up to four joints (eight Myorobotics actuators) at an update rate of 500 Hz. By using the full Myorobotics electronics, namely up to six MYO-Ganglia connected to a dedicated FlexRay controller, up to 12 joints (24 actuators) can be used at the same update rate. Again, with multiple SpiNN-IO boards, each with a dedicated FlexRay controller, we can alleviate this limit.

## Discussion

By demonstrating the control of a musculoskeletal joint with a simulated cerebellum running in real time, we successfully combined robotic hardware (Myorobotics) and simulation platform (SpiNNaker). Both Myorobotics and SpiNNaker offer scalability and usability: They can be extended in a straightforward manner, with no major roadblocks in sight toward systems approaching human-level complexity. Of course, many components still have to be added to arrive at a system that can interact with its environment in an intelligent way. Fortunately, a number of suitable technologies are readily available today.

### *Sensors*

While any sensor could be added to our framework, event-based systems are the most natural fit. Their sensory address-event-representation (AER) maps directly onto SpiNNaker packets, i.e., neural spikes in SNN simulations. The events emitted by an AER auditory sensor, or silicon cochlea [28], for instance, represent a sound's momentary frequency-resolved power spectrum. Their address encodes a specific frequency. The repetition rate of events with the same address encodes the respective spectral weight. Events emitted by an AER vision sensor, or silicon retina, typically represent a sudden, pixel-local change in brightness. Here, the address encodes the pixel coordinate. Silicon retinae [22],

**Practical and scalable systems like the one presented enable a mutual interaction between neuroscience and robotics: robots can help to advance neuroscience just as neuroscience helps us to create more natural robots.**

[29] and cochleae have previously been integrated with SpiNNaker. They are perfectly compatible with our interface.

### *Intelligence*

SpiNNaker can serve as a computing back end for PyNN [15] or Nengo [16]. Neural networks specified in those languages can often be run directly on SpiNNaker or require only minor modifications to be made, when porting a network from one compute back end to another. Missing software features, in our case a learning rule and I/O handlers, can be added to SpiNNaker's open source framework. Many available models can be ported and integrated into the system with minor effort.

### *Systems*

Neural models available for either PyNN or Nengo include diverse brain structures. In fact, the world's largest functional brain model, Spaun [30], is defined in Nengo. An embodied version of the model that can interact with the physical world as well as with humans would be an interesting test bed for human-robot interaction and cognitive science [31]. A Spaun-like brain model combined with advanced musculoskeletal robots like Roboy [3] (Figure 7) would herald a whole new era of robotic research. Our proof-of-concept system combining SpiNNaker and Myorobotics paves the way for exactly these kinds of endeavors, which we hope to stimulate with this article.

A typical human cerebellum comprises about 100 billion neurons [13], about as many as the rest of the brain. A realistic simulation of such a complex large-scale system will rely not just on massive computing resources; it also requires a detailed and realistic environment to interact with. Therefore, real-time capable neurosimulators in conjunction with robots will eventually become an essential tool of brain research. Practical and scalable systems like the one presented enable a mutual interaction between neuroscience and robotics: robots can help to advance neuroscience just as neuroscience helps us to create more natural robots.

## Acknowledgments

We wish to thank N. Luque for helpful discussions regarding cerebellar motor control; S. Temple and the SpiNNaker Manchester team for their invaluable hardware, software, and support; and the Myorobotics team for providing us with robot parts. Christoph Richter and Jörg Conradt acknowledge funding and support by the German Federal Ministry for Education and Research through the Bernstein Center for Computational Neuroscience Munich (01GQ1004A). Sören Jentzsch, Rafael Hostettler, Alois Knoll, Florian Röhrbein, and Patrick van der Smagt acknowledge funding from the European Union Seventh Framework Program (FP7/2007-2013) under grant agreement 604102 (Human Brain Project) and Rafael Hostettler under grant agreement 288219 (Myorobotics). Patrick van der Smagt also acknowledges support from DLR. Jesús A. Garrido and Eduardo Ros would like to acknowledge Spanish National Project NEUROPACT (TIN2013-47069-P). Jesús A. Garrido also acknowledges funding from the University of Granada and the European Union H2020 Framework Program (H2020-MSCA-IF-2014) under grant agreement 653019 (CEREBSENSING).

## References

[1] A. Bicchi and G. Tonietti, "Fast and 'soft-arm' tactics [robot arm design]," *IEEE Robot. Automat. Mag.*, vol. 11, no. 2, pp. 22–33, June 2004.

[2] O. Holland and R. Knight, "The anthropomimetic principle," in *Proc. AISB06 Symp. Biologically Inspired Robotics*, Bristol, UK, 2006, pp. 115–123.

[3] R. Pfeifer, H. G. Marques, and F. Iida, "Soft robotics: The next generation of intelligent machines," in *Proc. 23rd Int. Joint Conf. Artificial Intelligence (IJCAI-13)*, Beijing, China, 2013, pp. 5–11.

[4] S. Wittmeier, C. Alessandro, N. Bascarevic, K. Dalamagkidis, D. Devereux, A. Diamond, M. Jäntsch, K. Jovanovic, R. Knight, H. G. Marques, P. Milosavljevic, B. Mitra, B. Svetozarevic, V. Potkonjak, R. Pfeifer, A. Knoll, and O. Holland, "Toward anthropomimetic robotics: Development, simulation, and control of a musculoskeletal torso," *Artificial Life*, vol. 19, no. 1, pp. 171–193, Jan. 2013.

[5] H. G. Marques, M. Christophe, A. Lenz, K. Dalamagkidis, U. Culha, M. Siee, P. Bremner, and the Myorobotics Project Team, "Myorobotics: A modular toolkit for legged locomotion research using musculoskeletal designs," in *Proc. 6th Int. Symp. Adaptive Motion of Animals and Machines (AMAM'13)*, Darmstadt, Germany, 2013, pp. 10–15.

[6] J. Conradt, G. Tevatia, S. Vijayakumar, and S. Schaal, "On-line learning for humanoid robot systems," in *Proc. Int. Conf. Machine Learning (ICML2000),* Stanford, CA, 2000, pp. 191–198.

[7] F. Naveros, N. Luque, J. Garrido, R. Carrillo, M. Anguita, and E. Ros, "A spiking neural simulator integrating event-driven and time-driven computation schemes using parallel CPU-GPU co-processing: A case study," *IEEE Trans Neural Netw. Learn. Syst.*, vol. 26, no. 7, pp. 1567–1574, July 2015.

[8] T. Yamazaki and J. Igarashi, "Realtime cerebellum: A large-scale spiking network model of the cerebellum that runs in real time using a graphics processing unit," *Neural Netw.*, vol. 47, pp. 103–111, Nov. 2013.

[9] E. Ros, E. Ortigosa, R. Carrillo, and M. Arnold, "Real-time computing platform for spiking neurons (RT-spike)," *IEEE Trans. Neural Networks,* vol. 17, no. 4, pp. 1050–1063, July 2006.

[10] R. Agis, E. Ros, J. Diaz, R. Carrillo, and E. M. Ortigosa, "Hardware event-driven simulation engine for spiking neural networks," *Int. J. Electronics,* vol. 94, no. 5, pp. 469–480, Apr. 2007.

[11] J. Schemmel, D. Bruderle, A. Grubl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. 2010 IEEE International Symp. Circuits and Systems (ISCAS),* Paris, France, pp. 1947–1950.
[12] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Sci.*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.
[13] B. B. Andersen, L. Korbo, and B. Pakkenberg, "A quantitative study of the human cerebellum with unbiased stereological techniques," *J. Comp. Neurol.*, vol. 326, no. 4, pp. 549–560, 1992.
[14] S. Furber, F. Galluppi, S. Temple, and L. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.
[15] A. P. Davison, D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger. (2009, Jan.). PyNN: A common interface for neuronal network simulators. *Front. Neuroinformatics* [Online]. 2, p. 11. Available: http://www.frontiersin.org/neuroinformatics/10.3389/neuro.11.011.2008/abstract
[16] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. R. Voelker, and C. Eliasmith. (2014, Jan.). Nengo: A Python tool for building large-scale functional brain models. *Front. Neuroinformatics* [Online]. 7, p. 48. Available: http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2013.00048/abstract
[17] K. Doya, "Complementary roles of basal ganglia and cerebellum in learning and motor control," *Curr. Opin. Neurobiol.*, vol. 10, no. 6, pp. 732–739, 2000.
[18] P. van der Smagt, G. Metta, and M. A. Arbib, "Neurorobotics: From sensing to action," in *Springer Handbook of Robotics*, 2nd ed. New York: Springer-Verlag, 2016.
[19] G. Holmes, "The cerebellum of man," *Brain*, vol. 62, no. 1, 1939.
[20] N. R. Luque, J. A. Garrido, R. C. Carrillo, O. J.-M. D. Coenen, and E. Ros, "Cerebellar input configuration toward object model abstraction in manipulation tasks," *IEEE Trans. Neural Networks*, vol. 22, no. 8, pp. 1321–1328, Aug. 2011.
[21] D. Bullock and J. Contreras-Vidal, "How spinal neural networks reduce discrepancies between motor intention and motor realization," in *Variability and Motor Control,* K. Newell and D. Corcos, Eds. Champaign, IL: Human Kinetics Press, 1993, pp. 183–221.
[22] C. Denk, F. Llobet-Blandino, F. Galluppi, L. A. Plana, S. Furber, and J. Conradt, "Real-time interface board for closed-loop robotic tasks on the SpiNNaker neural computing system," in *23rd Int. Conf. Artificial Neural Networks (ICANN),* Sofia, Bulgaria, 2013, pp. 467–474.
[23] D. Marr, "A theory of cerebellar cortex," *J. Physiol.,* vol. 202, no. 2, pp. 437–470, June 1969.
[24] J. S. Albus, "A theory of cerebellar function," *Math. Biosci.*, vol. 10, no. 1–2, pp. 25–61, Feb. 1971.
[25] Y. Yang and S. G. Lisberger, "Purkinje-cell plasticity and cerebellar motor learning are graded by complex-spike duration," *Nature*, vol. 510, no. 7506, pp. 529–532, June 2014.
[26] N. R. Luque, J. A. Garrido, R. R. Carrillo, O. J.-M. D. Coenen, and E. Ros, "Cerebellarlike corrective model inference engine for manipulation tasks," *IEEE Trans. Syst. Man Cybern. B*, vol. 41, no. 5, pp. 1299–1312, Oct. 2011.
[27] E. Ros, R. Carrillo, E. M. Ortigosa, B. Barbour, and R. Agís, "Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics," *Neural Computation*, vol. 18, no. 12, pp. 2959–2993, 2006.
[28] V. Chan, S. C. Liu, and A. van Schaik, "AER EAR: A matched silicon cochlea pair with address event representation interface," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 1, pp. 48–59, 2007.
[29] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 × 128 120 db 15 μs latency asynchronous temporal contrast vision sensor," *IEEE J. Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, Feb. 2008.
[30] C. Eliasmith, T. C. Stewart, X. Choo, T. Bekolay, T. DeWolf, Y. Tang, and D. Rasmussen, "A large-scale model of the functioning brain," *Sci.*, vol. 338, no. 6111, pp. 1202–1205, Nov. 2012.
[31] J. L. Krichmar, "Design principles for biologically inspired cognitive robotics," *Biologically Inspired Cognitive Architectures*, vol. 1, pp. 73–81, 2012.

***Christoph Richter***, Bernstein Center for Computational Neuroscience Munich and Neuroscientific System Theory, Department of Electrical and Computer Engineering, Technische Universität Munich, Germany. E-mail: c.richter@tum.de.

***Sören Jentzsch***, fortiss GmbH, Associate Institute of the Technische Universität Munich, Germany. E-mail: soren.jentzsch@gmail.com.

***Rafael Hostettler***, Robotics and Embedded Systems, Department of Informatics, Technische Universität Munich, Germany. E-mail: rh@roboy.org.

***Jesús A. Garrido***, Department of Computer Architecture and Technology, Information and Communication Technologies Research Center, University of Granada, Spain. E-mail: jesusgarrido@ugr.es.

***Eduardo Ros***, Department of Computer Architecture and Technology, Information and Communication Technologies Research Center, University of Granada, Spain. E-mail: eros@ugr.es.

***Alois Knoll***, Robotics and Embedded Systems, Department of Informatics, Technische Universität Munich, Germany. E-mail: knoll@in.tum.de.

***Florian Röhrbein***, Robotics and Embedded Systems, Department of Informatics, Technische Universität Munich, Germany. E-mail: florian.roehrbein@in.tum.de.

***Patrick van der Smagt***, fortiss GmbH, Associate Institute of the Technische Universität Munich and Robotics and Embedded Systems, Department of Informatics, Technische Universität Munich, Germany. E-mail: smagt@tum.de.

***Jörg Conradt***, Neuroscientific System Theory, Department of Electrical and Computer Engineering, Technische Universität Munich, Germany. E-mail: conradt@tum.de.