Integrated neural and robotic simulations. Simulation of cerebellar neurobiological substrate for an object-oriented dynamic model abstraction process

Niceto R. Luque [a,*,1], Richard R. Carrillo [a,1], Francisco Naveros [a], Jesús A. Garrido [b], M.J. Sáez-Lara [c]

[a] *Department of Computer Architecture and Technology, CITIC, University of Granada, Periodista D. Saucedo Aranda s/n, E-18071 Granada, Spain*

[b] *Brain Connectivity Center, IRCCS Istituto Neurologico Nazionale C. Mondino, Via Mondino 2, Pavia, I-27100, Italy*

[c] *Department of Biochemistry and Molecular Biology, University of Granada, CIBM, PTS, s/n, E-18071 Granada, Spain*

This document is a preprinted version of the final manuscript published in ROBOTICS AND AUTONOMOUS SYSTEMS. The original reference is "Luque, N. R., Carrillo, R. R., Naveros, F., Garrido, J. A., & Sáez-Lara, M. J. (2014). Integrated neural and robotic simulations. Simulation of cerebellar neurobiological substrate for an object-oriented dynamic model abstraction process. *Robotics and Autonomous Systems*, *62*(12), 1702-1716". The DOI for the original document is: https://doi.org/10.1016/j.robot.2014.08.002

HIGHLIGHTS

- We integrated EDLUT neural simulator within a simulated robotic environment.
- As an embodiment example, we implemented a cerebelar-like structure controlling a simulated arm.
- The neural robotic simulator combines signals in analog/spike domains.
- Neural simulator, interface, and robotic platform operate conjointly in real time.

ABSTRACT

Experimental studies of the Central Nervous System (CNS) at multiple organization levels aim at understanding how information is represented and processed by the brain's neurobiological substrate. The information processed within different neural subsystems is *neurocomputed* using distributed and dynamic patterns of neural activity. These emerging patterns can be hardly understood by merely taking into account individual cell activities. Studying how these patterns are elicited in the CNS under specific behavioral tasks has become a groundbreaking research topic in system neuroscience. This methodology of synthetic behavioral experimentation is also motivated by the concept of embodied neuroscience, according to which the primary goal of the CNS is to solve/facilitate the body–environment interaction.

With the aim to bridge the gap between system neuroscience and biological control, this paper presents how the CNS neural structures can be connected/integrated within a body agent; in particular, an efficient neural simulator based on EDLUT (Ros et al., 2006) has been integrated within a simulated robotic environment to facilitate the implementation of object manipulating closed loop experiments (action–perception loop). This kind of experiment allows the study of the neural abstraction process of dynamic models that occurs within our neural structures when manipulating objects.

The neural simulator, communication interfaces, and a robot platform have been efficiently integrated enabling real time simulations. The cerebellum is thought to play a crucial role in human-body interaction with a primary function related to motor control

which makes it the perfect candidate to start building an embodied nervous system as illustrated in the simulations performed in this work.

***keywords**:* Neurobotics, Cerebellum, Spiking neural network, Close-loop simulation, Embodied neuroscience

1. Introduction

Computational models of various brain regions have been developed and studied for more than thirty years in order to analyze central brain functions. Computational neuroscience (CN) is the natural complement of experimental brain research, since it focuses on specific mechanisms and models which are only partially observed in anatomical or physiological studies. In particular, the cerebro-cerebellar loop has been extensively modeled since Marr and Albus [1,2], providing elegant explanations on how the forward controller operation of the cerebro-cerebellar loop seems to work. Nevertheless, these computational theories tend to focus on one part of the cerebellar circuitry and then, to extrapolate the obtained conclusions to the whole cerebro-cerebellar system. Simulating nervous systems ''connected'' to a body (agent or robot with sensors and actuators) is of interest for studying how certain capabilities of the nervous system (e.g. the role of the cerebellum in coordinated movements and object manipulation) are based on cellular characteristics, nervous system topology, or local synaptic adaptation mechanisms. This represents an integrative approach which aims to build the bridge between task specific experimentation (equivalent to ''awake animal testing'') and system neuroscience models.

This integrative approach allows us to study the role of certain nervous systems within ''behavioral tasks'' [3]. For this purpose, it is crucial to study nervous system models within the framework of their interaction with a body (sensors and actuators) and the environment.

This paper describes an integrated approach to the cerebellar circuit modeling within real time ''behavioral tasks''. The paper describes briefly: (a) a cerebellar model based on point neurons capable of being simulated in real-time. The model maintains biological interconnectivity ratios in functional medium-scale networks (rather than an ad-hoc neural network particularly designed for a specific behavioral task) that are embedded in biologically plausible control loops. (b) Testing the role of plasticity at parallel fibers-Purkinje cells. (c) Embedding the neural system model into a cerebro-cerebellar control loop connected to a Light Weight Robot (LWR) performing repetitive fast manipulations along benchmark trajectories. In order to address these three aims, we have integrated a neural simulator based on EDLUT [4] with a simulated robotic environment to facilitate the implementation of object-manipulating closed-loop experiments (action–perception closed loops).

These experiments allow us to study the neural abstraction process of dynamic models (of objects being manipulated) that occurs within our neural structures in fast manipulation tasks [5–7]. The neural simulator, the communication interface, and the simulated robotic platform have been developed and integrated taking into account computational efficiency as a major requirement in order to enable real time simulations. This platform allows us to study different neural representation and processing schemes in a specific task within a brain–body interaction framework.

1.1. Functional cerebellar models; a brief overview

Among Embodied System Neuroscience models, the well-organized structure of the cerebellum has received special attention from researchers belonging to very

different fields. On one hand, neurophysiologists have studied and proposed detailed models and descriptions according to experimentally recorded cells and synaptic properties. However, it is not yet clear how specific properties of these current detailed models facilitate specific tasks at a behavioral level. On the other hand, engineers have proposed artificial approaches (only related with biology at a very high level) for biologically relevant tasks such as accurate and coordinated movements. Based on these opposed approaches, several cerebellar modeling frameworks have been proposed:

In *state-generator models*, the granule cell layer presents on/off type ''granule'' entities that provide a sparse coding of the state space (Marr–Albus Model [1,2], CMAC [8–10] model, or Yamazaki and Tanaka model [11–14]). These models succeed in explaining some traditional cerebellum-involving tasks such as eyelid conditioning [15] or motor control tasks [6,7,16]. In *functional models*, only the functional abstraction of specific cerebellar operations is considered: MPFIM model [17], Adaptive Filter model [18–22], APG model [23], or LWPR model [24,25]. Although in some cases, these models are also used to explain how the cerebellum works, these can be seen as problem solving approaches (that use internal structures not constrained to biologically plausible features). These functional models are also used to study the potential role of the cerebellum in tasks such as eyelid conditioning, the vestibule ocular reflex (VOR), or movement correction [24,25]. Finally, *cellular-level models* capture the biophysical features of the cerebellar neuronal topology and processing, and can be evaluated in the framework of neurophysiological experiments. These models aim to be as biologically plausible as possible. But due to their inherent complexity, their application in the context of large-scale cerebellar modeling and computation remains limited. The very first approximations in this field were developed based on the simplified models of Schweighofer–Arbib [26,27].

1.2. How to embody the cerebellar circuitry

The cerebellar network has been at the core of neurocomputational theories since the 1960s, when Eccles proposed the Beam Theory [28] and Marr and Albus, the Motor Learning Theory [1,2]. Later on, Ito developed the forward controller theory [4,29–32]. Since then, the view has been crystallized on two main concepts that can be synthesized as follows; the way the cerebellum operates is by decorrelating the inputs in the granular layer and detecting known patterns in Purkinje cells. Pattern recognition is regulated by memory storage at the parallel-fiber-Purkinje-cell synapse. When unfamiliar patterns are detected repeatedly, the Purkinje cells change their firing rate and regulate activity in the deep cerebellar nuclei (DCN), thereby emitting the corrective terms used for highly accurate motions (skillful control performance).

Despite its attractiveness and simplicity, this theory only partially accounts for the capabilities of the cerebellum. Furthermore, recent experimental data indicate that the cerebellar system is much more complex than initially stated. Just to make a very short survey, the mechanisms of the granular layer go far beyond simple decorrelation [33], long-term synaptic plasticity does not occur only at the parallel fibers (PF) [33–35], the inferior olive (IO) operates as a complex timing system and not simply to drive Purkinje cell plasticity [36], the Purkinje cells and the DCN cells have operative states that go far beyond the concept of firing rate regulation [37]. The core idea is that our knowledge on the functioning of neuronal networks of the cerebellum is still rather vague, and that we have to develop new computational tools to investigate cerebellar network dynamics beyond the current existing paradigms.

The available neurophysiological data (which is essential for understanding the functional organization of the cerebellum and related structures) has to be analyzed to

investigate the particular processing capabilities of each neuron and of its internal dynamics. Emphasis must be put on proving how the network processing capabilities are supported by the low-level characteristics of each neuron type. Many of the specific cerebellar neural types have already been implemented in Python–NEURON–EDLUT software simulators [38,39] and there are even specific repositories gathering different kinds of models [40,41].

1.3. Modeling the cerebellar circuits

When modeling the cerebellar circuit with a bottom-up approach, the cerebellar network needs to be modeled aiming at the construction and generation of a complete cerebellar functional network, tested in realistic functional conditions and endowed with plasticity rules. This process demands the comprehension of the interplay that occurs between the Granular-and-molecularlayer subcircuit and the PC–DCN–IO subcircuit.

Whilst the granular layer and molecular layer neurons can be largely reconstructed starting from precise existing models, the DCN-and-IO subcircuits are not modeled in detail. Therefore, the PC–DCN–IO circuit requires basic modeling to achieve functional properties. An initial model of the DCN can be constructed based on [42,43]. As a starting point, the IO can be modeled at a functional level, i.e. as a module translating ''error related signals'' into activity that modulates learning at the PF–PC synapses. Also at this stage, although different plasticity sites have been reported [34,35], most cerebellar functional models are based solely on the PF–PC adaptation mechanism modulated by the IO activity (which delivers the teaching signals).

Once all the subcircuits and long-term synaptic plasticity are implemented and tested separately, the functional operation of a complete circuit can be tested. The first step lies on developing an appropriate connectivity between the modular subcircuits. The connection map between the IO and PCs via climbing fibers, the convergence of PCs to DCN neurons [44] and the mossy fiber (MF) projections to the DCN [45], and the granular layer have been extensively described in the literature and should be reconstructed respecting the known convergence/divergence ratios.

2. Material and methods

2.1. The real-time neural simulator. EDLUT

Common event-driven simulators [46,47] use simple neural models whose dynamics are described by equations which can be discontinuously evaluated at arbitrary times (e.g. current based integrate-and-fire models). But even when using simple neural models, the firing-time prediction which is necessary for an eventdriven simulation may be complex [48,49].

An EDLUT (Event-Driven neural simulator based on LookUp Tables) was implemented [4] to simulate neural models whose internal dynamics is defined by a set of differential equations (for instance, the Hodgkin and Huxley model [50]) adopting an event-driven simulation scheme. This software is an open source project [51] for efficient simulation of biological neural networks. It is of particular interest in the field of neurobotics and embedded neural computing in which real-time processing is required, for example, for experiments which include perception–action loops.

EDLUT uses an intensive preliminary simulation stage in which a neural model is characterized, i.e. massive simulations of a single cell are done with different initial conditions. At this stage, samples of the neural variables at different times are stored

in lookup tables. This preliminary stage can be seen as a cell model compilation stage. These tables are calculated using time-consuming numerical analysis (e.g. Runge–Kutta method). However, once they are generated, the network simulation can be run efficiently through the event-driven method, just by accessing tables when the neural state must be updated or predicted.

EDLUT uses lookup tables which store all the possible values (with certain precision) of the neural-model state variables [52] in addition to the future states (firing times) [53]. Therefore, a whole neural model is encoded in each set of model-characterization tables. In this way, the simulator takes advantage of the increasing memory resources available to perform efficient simulations with very limited computation requirements. The event-driven simulation scheme based on lookup tables uses memory access intensively, instead of CPU computation power for the neural variable updates.

The initial EDLUT processing scheme allowed fast simulation of complex neural models. Nonetheless, this scheme is constrained by the number of state variables of a neural model because this determines the number of dimensions of the required lookup tables. But in later versions [54], the EDLUT was upgraded to provide a hybrid time-and-event driven simulation method. This hybrid scheme allows the concurrent simulation of some neuronal models using the event-driven method (the models which can be translated into lookup tables) and other models using the time-driven method in the same network.

2.2. The cerebellar model

We have used leaky integrate-and-fire neural models (LIF) [50] whose synapses are modeled as conductances. The general model has been then adapted for different neural types. The LIF neural state is characterized by the membrane potential ($V_{m\text{-}c}$) expressed by Eq. (1):

$$C_m \frac{dV_{m-c}}{dt} = g_{AMPA}(t)(E_{AMPA} - V_{m-c}) + g_{GABA}(t)(E_{GABA} - V_{m-c}) + G_{rest}(t)(E_{rest} - V_{m-c}) \quad (1)$$

where C_m stands for the membrane capacitance, *EAMPA* and *EGABA* denote the reversal potentials of the synaptic conductances, and finally, E_{rest} represents the resting potential (being G_{rest} the conductance responsible for the passive decay term towards the resting potential). The g_{AMPA} and g_{GABA} conductances integrate all the contributions received through individual synapses and are defined as decaying exponential functions. The parameters of the neural model [5–7] and a more detailed description can be found in [5–7,51].

Therefore, the state of a neuron is defined with just three variables:

$V_{m\text{-}c}$ represents the membrane potential. When this variable reaches a specific threshold, the neuron generates an output spike.

g_{AMPA} and g_{GABA} represent excitatory and inhibitory conductances respectively that affect the membrane potential. These conductances decrease exponentially in each integration step and increase proportionally to the synaptic weight of their connections when an input spike arrives.

To solve the LIF neuron model differential equation, the EDLUT simulator incorporates different integrative methods. This differential equation is processed off-line using a short integration step to achieve good accuracy (it does not directly affect the computation time during system neural simulations, since the neural model is computed and stored in lookup tables in a preliminary neural characterization stage).

All the different characterized neural types have been interconnected following a cerebellar topology structured into micro-zones distributed in different layers, as described below (Fig. 1):

Mossy fibers (MFs) (248). These mossy fibers drive the contextual information and sensory joint information (related with the manipulated object and desired/actual positions and velocities). The mossy fiber model is based on leaky integrate-and-fire neuron dynamics whose input current is provided by a set of overlapping receptive fields covering the joint value space of the input signals (see Fig. 2).

Granular layer (GCs) (1500). This layer behaves as an abstraction of a simplified cerebellar granular layer. The information provided by the mossy fibers is translated into a sparse representation. Each granular cell (GC) receives four excitatory input connections; three connections randomly chosen from joint-related mossy fibers and the other one, from a context-related mossy fiber [7].

Parallel fibers (PFs) (1500). They represent the output axons of the granular layer. The manipulated object model abstraction is stored in learned weights at the PF–PC connections.

Climbing fibers (CFs) (48). The climbing fibers are the axons of the Inferior Olive cells. This layer consists of 6 groups of 8 climbing fibers each. The IO output (encoding a teaching signal related to the error) is translated into spikes using leaky integrate-and-fire neuron dynamics whose input current is in this case proportional to the error signal. The CFs drive the IO outputs to the Purkinje cells for supervised learning at PF–PC connections. More details on this learning rule can be found in [6].

Purkinje cells (PC) (48). These cells are divided into 6 groups of 8 cells. Each GC is connected to 80% of the PCs which are also receiving their corresponding teaching signals from the CFs.

Deep cerebellar nuclei cells (DCN) (24). The cerebellar output is generated using 6 groups of these cells (2 groups per joint) whose activity is capable of providing corrective torques for a specified cerebellar input. Corrective torque values per joint are encoded by a couple of these groups, one group compensating positive errors (agonist) whilst the other one is dedicated to compensate negative errors (antagonist). Each DCN neuron group receives excitation from every MF cell and inhibition from the two corresponding PCs. The sub-circuit PC–DCN–IO then is organized into six microzones; three of them generating joint positive corrections (one per joint) and the other three, generating joint negative corrections (one per joint). Mind that as it will be explained below, we use three joints in our robot experiments.

2.3. Neural population coding

Neural population coding is traditionally used for sensorimotor representation. Each neuron belonging to a certain system presents a distribution of responses over some set of inputs. Hence, the response of many system neurons over a set of certain inputs represents the system state [55,56]. In a reaching movement, the arm direction is encoded by means of neurons whose input current changes with the cosine of the difference between the stimulus angle and the preferred direction of the cell [57] (*Cosine tuning*). Each cell has a preferred direction and receives input current depending on how a movement is aligned to its preferred feature. However, a simple reaching movement involves extracting spatial information including visual acquisition of the target, coordination of multi-modal proprioceptive signals, and a proper motor command generation to drive a proper motor response towards the target [58]. Common reaching movements towards a target that we have already seen involve an internal representation of the target and limb positions, and also a coordinate transformation between different internal reference frames. A spiking population

coding is used as internal representation and can be adapted as indicated below to be embedded into a control loop.

The integration of computational models with neurophysiolog-

ical observations in order to understand the main problems in motor control requires not only the cerebellum functionality to be considered but also its biological architecture (cell-network topology). This requires the development of two ''translation processes'' in order to interact with a robot agent: (1) Translation from analog domain sensor inputs to spike based patterns compatible with a spiking cerebellar network. (2) Translation from spike domain cerebellar outputs to analog domain actuator commands to be delivered to the robot agent.

2.3.1. From sensors to spikes

When a target reaching movement is executed, different body parts, such as muscles, tendons, or joints are articulated depending on their body location [59] along the followed trajectory. Sensory proprioceptors are activated according to the movement; thus, a time-varying set of stimuli is produced, and its corresponding neural population varying activity is generated. In contrast, in a robot scenario, the only available proprioception sensory information is supplied by an encoder output per link. Hence, a translation from the joint position/velocity measures to a time-varying set of stimuli is required. At this point, finding out an optimal biologically plausible encoding scheme that allows ''biological decoders'' (as the ones we assume at the granular and molecular layers of the cerebellum) to take advantage of the representation is a non-trivial issue. It is assumed that the firing rate of an individual sensory receptor follows a neural response which is characterized by Eq. (2) (also equivalent to a *cosine tuning* curve, that is, the firing rate of the neurons varies with the angle between the preferred direction of the sensory receptors and the sensed position) [60]. Therefore, a reaching movement execution will be represented with a sparse population of active cells which are varying with time. This coding mechanism leads to a representation of the current sensorial state during the trajectory execution in an unambiguous way.

The output of each receptor is given by Eq. (2);

$$I_{Ni}(t) = r_0 + r_{max} \sum_n e^{-\frac{(\theta - \theta_{pref} - 2\pi n)^2}{2\sigma^2}} \tag{2}$$

where [r_0, $r_{\max}$] is the joint range in radians, ϑ is the actual position, ϑ_{pref} is the preferred direction of the receptor, σ is the amplitude of the receptive field associated to the receptor, and finally, $2\pi n$ is a subtractive term used to refer the actual position to the first-360-degrees (the maximal range of any revolute joint is 360°).

Receptors are distributed along the range of each joint, being their receptive fields overlapped (as peripheral nerve receptive fields are). Each value of a proprioceptor output signal is integrated using an integrate-and-fire neuron model whose dynamics is defined in Eq. (3) (see illustration in Fig. 2). In the case of an arm system, this determines the output activity that drives the Cuneate Nucleus (CN) activity emulating the way the Mossy Fiber activity from cells in the CN handles information from forelimb muscle spindles [61].

$$\tau_{mi} \frac{dv_i}{dt} = -v_i(t) + R_i I_{Ni} \tag{3}$$

Related to the leakage integrate and fire cell dynamics, τ_{mi} is the resting time constant, v_i the membrane potential, I_{Ni} the input current, and R_i is related to the resting conductance of the membrane.

2.3.2. From spikes to actuators, decoding the cerebellar output

Spiking modeled neurons elicit pulsed signals usually named action potentials or spikes. It is believed that the shape of these spikes only carries minimal information whilst the core of the information is carried by the spike time arrival [62,63]. The action potential waveforms (voltage curve profile) elicited by those neurons is usually translated into a set of binary symbols (0 or 1) representing an instant in which an action potential occurs (1) or does not

(0). The generated binary waveform conforms a spike train and the obtained pattern of spikes belonging to a certain time-frame generates the spike binary code; the columns corresponding to the array of spikes are also named *neural activation patterns*. It is then clear that, somehow, the translation of these neural activation patterns into meaningful analog output signals has to be implemented for interfacing actual robot actuators with analog signals.

Assuming that the goal is to decode rather than to analyze the behavior of biological neurons, it seems reasonable to use a mathematical approach such as linear filtering, particularly, a Finite Impulse Response filter (FIR), to accomplish this task [64].

Defining the spike train as $x(t) = \sum_{j=t}^{N} \delta(t - t_j)$, where t_j stands for the set of firing times of the corresponding neuron and being the FIR response defined as *h(t)*, then the stimulus can be written as follows:

$$stimulus(t) = (h * x)(t) = \sum_{j=t}^{N} h(t - t_j) \tag{4}$$

As noticed from Eq. (4), converting spike trains into analog signals is a quite straightforward implementation. Nevertheless, despite the widespread use of FIR filters for such purpose, an undesired delay is introduced in the generated analog signal. This delay is strongly related to the number of filter coefficients as well as to the shape of the filter kernel. To mitigate this effect and to make the conversion more efficient, an exponentially-decaying kernel can be implemented, as seen in Eq. (5). Thus, at each time step, the output signal value only depends on its previous value and on the input spikes in the same time step. Therefore, this filter can be implemented by recursively updating the last value of the output signal. Actually, the choice of such exponential kernel is double folded. The kernel is able to mitigate the delay problem and bears a strong resemblance to postsynaptic currents [62,63], thus facilitating a possible biological interpretation.

$$Kernel = h(t) = e^{-\frac{M}{\tau}} \tag{5}$$

where *M* is the number of filter taps (one per integration step) and
τ is the decay factor.

2.3.3. Equivalent to an integrative neuron

Integrative neurons are capable of both analyzing and interpreting sensory input just taking into account their actual state, the incoming information, and their previous states as well. Once the computation of those three elements is done, the resulting information can be transmitted to motor neurons or other integrative neurons. Assuming a leaky integrate-and-fire model for the integrative neuron, the model looks like Eq. (3). This model forces the

input current to exceed a threshold $I_{th} = V_{th}/R_i$ for the cell i to fire; otherwise, it will simply leak out any charge in the membrane potential. The firing frequency is thus defined in Eq. (6):

$$f(I) = \begin{cases} 0, & if\ I_{Ni} \leq I_{th} \\ \left(t_{ref} - \tau_{mi} log\left(1 - \frac{V_{th}}{I_{Ni} R_i}\right)\right)^{-1}, & if\ I_{Ni} > I_{th} \end{cases} \quad (6)$$

where t_{ref} is a refractory period and τ_{mi} is the resting time constant. Solving the differential equation (3), the membrane potential is expressed as follows:

$$v_i(t) = R_i I_{Ni}(t) + \frac{V_{rest} - R_i I_{Ni}(t)}{e^{(t/\tau_{mi})}} \quad (7)$$

The functionality of the selected FIR described in Eqs. (4) and (5) can be read in terms of a biological interpretation just by making an analogy between the proposed exponential-decaying kernel and the behavior of an integrative neuron whose dynamics is de fined using Eq. (7) (stimulus$(t) \approx V_j(t)$). The resulting shapes of both sides of this analogy hold a remarkable resemblance due to the exponential-decaying kernel that governs both the neural dynamics and the FIR kernel. An engineering strategy usually adopts the FIR based approach, because it allows us to easily adapt the output values to the control signal which is demanded for accurate control. In such a way, the effect of each spike elicited by any cerebellar nuclei cell (output cerebellar cells) can be easily pondered thanks to the FIR filter, thus facilitating the correlation between the cerebellar output spikes and their corresponding corrective output signals. It is clear then that this conversion can be also processed by using any Integrate and Fire-like neuron; however, doing so, the influence of each spike on the output does not always remain clear.

2.4. Cerebellar control loop; a plausible implementation

It is widely assumed that the cerebellum, acting as a control module, is embedded in a feedforward control loop [65–67]. A feedforward control system is able to evaluate both the incoming sensory information from the environment and the information provided by the system itself (proprioception) before the motor control action is sent to the body. This means that the controller manages the sensory information to deliver the best motor commands to accomplish the desired movement. At that point, we must bear in mind that once a pure feedforward system sends the corresponding control actions, it is not possible to modify them.

On one hand, a feedforward control system is able to deliver the precise set of motor commands for the body-plant and to make corrections during the movement without continuously checking the motor control output [26,27]. Conversely, the feedforward controller requires a previous trial-and-error learning process in order to later recognize (in a recall stage) all the possible sensorial states that may be reached. In a real manipulation task, the environmental conditions are constantly changing and the feedforward con troller must continuously tune its motor commands to cope with these changeable environmental conditions [68]. According to this scheme, the cerebellum operates as a feedforward controller for the motor commands which are originated in the motor cortex (Fig. 3). The brain is able to plan and learn the optimal trajectory of a movement in intrinsic coordinates [23,68–71]. This operation consists of three main tasks: the desired trajectory computation in external coordinates, the task-space translation into body coordinates, and the motor command generation [72]. In order to deal with the aforementioned changeable environmental conditions, the

system needs to incorporate a Feedback-Error Learning (FEL) scheme [73] by means of the cerebellum operating in conjunction with a crude inverse dynamic model of the arm-plant [74]. It has been proposed that the association cortex provides the motor cortex with the desired trajectory in body coordinates. In the motor cortex, the motor command is calculated by using an inverse dynamic arm model (for a review, see [75]). The *spinocerebellum–magnocellular red nucleus* system provides an accurate model of musculoskeletal dynamics, which is learned with practice by sensing the motor command consequences in terms of executed movements (proprioception). The *cerebrocerebellum–parvocellular red nucleus system*, which projects back to the motor cortex, provides a crude inverse-dynamic model of the musculoskeletal system, which is acquired whilst monitoring the desired trajectory [73]. The crude inverse-dynamic model works together with the dynamic model provided by the cerebellum embedded in a feedforward control loop thus updating motor commands accord ing to predictable errors occurring when executing a movement. It learns and stores models of the skeleto-muscular system providing the precise timing control of agonist–antagonist muscle pair groups in addition to the needed force and stiffness control [76]. Obviously, the muscle flexion–contraction precise timing and the needed force in a manipulation task depend on the weight to be handled (more concretely, on the dynamic model of the object under manipulation), the cerebellum being crucial for delivering this proper timing, force, and coordination; these appropriate corrective terms are learned through a trial-and-error process [68].

2.5. Simulated robot integration: robot and training trajectory

Behavioral experiments with an embodied cerebellar system require the integration of a real or simulated robot in the control loop. The simulated robot is intended to follow a specific trajectory whilst the cerebellar model learns to provide corrective torques for the robot actuators. The robot-control experiment results are intended to assess the effects on performance caused by concrete neural properties, cerebellar subcircuits, or adaptive mechanisms (synaptic plasticity). This robot-control experimentation demands human-like robots whose intrinsic dynamics is somewhat similar to their biological counterparts. This requirement motivates the use of lightweight robots (LWR) such as the Kuka lightweight robot developed by DLR [77,78].

As mentioned above, the main role of the cerebellum seems to be related to human motor control, especially in those tasks where timing and force are critical. Therefore, those manipulation tasks able to modify the dynamics of the arm-plant whilst performing certain movements would constitute the paradigm to follow. These LWR robots are capable of being dynamically modified when manipulating different payload contexts under certain kind of movements. This motivates the definition of a benchmark trajectory capable of revealing the dynamic properties of a LWR. According to the proposals in [76,79], fast movements in a smooth pursuit task consisting of vertical and horizontal sinusoidal components are good candidates in order to reveal the robot dynamics. Examples of different benchmark trajectories can be checked in [74,76,80]. Considerations related to the communication interface delay and the friction force of the robot joints need to be taken into account (see Appendix).

2.6. The integrated neurobotics simulation platform

These techniques are now included into an integrated software platform able to combine realistic robotic experiments (running in real time) with cerebellar like modules that work as corrective engines. This platform aims to facilitate the study of how the adaptive neural information coding mechanisms underlying the ability of

humans to interact with their environment is handled by means of an effective adaptation at the cerebellum. The simulator of the robotic LWR arm, the control loop, and the cerebellar module were implemented in C/C++ following previous developments [5–7,24,25]. The software platform source code has been made available at: https://code.google.com/p/edlut/source/browse/branches/EDLUT_with_Robot.

The core of the neural simulator was implemented taking EDLUT [4] source code as the basis. EDLUT was then provided with an *interface library* as well as with a *robot library* able to dynamically define and model different lightweight robot configurations. In this work, we use a rough approximation of a Kuka LWR [77].

2.7. A practical running example

The aim of this working example is to show how a cerebellar model based on [7] within a ''perception–action'' closed-loop [5–7,74] is used in order to control a simulated LWR [77] arm by means of the developed software platform. Vertical and horizontal sinusoidal composed trajectory-following tasks [5–7,74] will be run in order to reveal the robot dynamics (see Fig. 3) with different payloads to be manipulated. The input pathways to the artificial cerebellum will be MFs and CFs. The cerebellar output is translated into torque commands for each joint through conversion modules [5–7], following the approach described in the previous section.

3. Results

Using this cerebellar architecture, a 1000 trial execution (each trial takes one second) following the principles already presented in [7] has been performed, obtaining the raster plot shown in Fig. 4. This figure represents a snapshot of one trial execution representing a cerebellar simulation of one second eight-like trajectory operating 3 revolute joints (joint 1, joint 2, and joint 3 indicated in Fig. 3) of a LWR defined in [5–7] when manipulating a 10 kg payload. This snapshot corresponds to two particular moments during the learning process; the initial learning stage (left column plots) and the final learning stage (right column plots). Mind that, as can be seen in Fig. 4, at the initial learning stage (0–1 s period), no cerebellar action has been learned yet (Fig. 4(E)), whilst at the final learning stage (999–1000 s period), the learning process is well settled down (Fig. 4(F)) and corrective terms are delivered through DCNs.

As we see, all plots represent activity along time using dots (in plots A and B, each dot represents a spike) or short vertical markers when the number of neurons being monitored is lower (plots C, D, E, and F). Fig. 4(A) represents a raster plot of the input activity that is reaching the cerebellar architecture through mossy fibers at the initial learning stage. As explained before, mossy fibers are able to elicit a set of spike trains related to the desired and actual positions and velocities (according to the scheme illustrated in Fig. 2) presented by the robot arm along the eight-like trajectory movement. Each joint position and velocity is translated into spikes by using three groups (one for position and another one for velocity for each joint) of 20 mossy fibers. Each of these groups is activated by its corresponding set of receptive fields (Fig. 2) that are covering the operative range of the input variable. At this initial learning stage, the actual trajectory is far from the desired one, thus position/velocity values only activate part of the population of mossy fibers (compared to the activation of the mossy fibers encoding the desired trajectory). However, as Fig. 4(B) shows, at the final learning stage, both actual position/velocity values can properly cover the operative range of the input variables. It can be seen that at this final learning stage, the activation of the mossy fibers related to the desired trajectory is similar to the activation profile of the

mossy fiber group related to the actual trajectory (encoding actual position and velocity along the movement execution).

The activity of mossy fibers reaches the granular cell layer. The granular layer operates adopting the model functionality described in [13–15] by Yamazaki and Tanaka, that is, it behaves as a state generator. A state generator machine is capable of representing each time step (in our simulations, this is 0.002 s) as an unambiguous time stamp (with a unique spike pattern representation), thus facilitating the learning process (see [7]).

As indicated in the description of the cerebellar architecture, the Purkinje cell activity is divided into 6 well-defined sets of spike trains representing the generated spiking activity related to the output agonist/antagonist joint micro complexes for each robot joint (joint 1, joint 2, and joint 3). Each pair of these 6 well-defined sets is related to each agonist/antagonist corrective action for the three joints. As aforementioned, the inferior olive activity (spiking patterns Fig. 4(C), (D)) is in charge of encoding the error signal (Fig. 4(C), (D) colored lines) that has to be compensated by the cerebellar corrective terms; here, we can see that there are also 6 well-defined areas related to micro-complexes encoding the positive/negative corrective actions for the three robot joints. Fig. 4(C),

(D) illustrates how the inferior olive spike distribution during the trajectory execution remains proportional to the received error signals which in turn, are related to the actual position/velocity errors. In these figures, it is shown how just a positive corrective action is demanded in joints 2 and 3 whilst both positive and negative actions are demanded in joint 1 along the whole eight-like trajectory execution. The error directionality (either positive or negative error) is also illustrated in Fig. 4(C) and (D). Obviously, at the initial learning stage, the amplitude of the encoded error signal to be translated into spikes is high as well as the number of spikes elicited by the inferior olive since the learning process has barely started (Fig. 4(C)). On the contrary, once the learning process is well settled down, the expected amplitude of the encoded error signal to be translated into spikes and the numbers of elicited spikes by the inferior olive decreased significantly at this final learning stage (Fig. 4(D)). The Inferior Olive cell activity is constrained between 1 and 10 Hz, according to neurophysiological data [81].

Finally, DCN generated output activity is plotted in Fig. 4(E) and (F). At the beginning of the learning stage, a negligible cerebellar output is provided (Fig. 4(E)) whilst at the final learning stage (Fig. 4(F)), an appropriate cerebellar output corrective action is generated. Error corrections are accomplished by changes in the activity of PCs that, in turn, affect the activity of the DCN, which eventually is translated into analog torque correction signals (also plotted in Fig. 4(E) and (F), with continuous lines) following principles already presented in the previous section. Each group of 4 DCN cells encodes the positive or negative corrective term which is eventually translated into a joint corrective torque. The higher/lower the activity at each micro complex is, the higher/lower its corresponding corrective torque is. In fact, the final activity at the DCN (which represents the actual corrective terms being produced) is the result of the subtraction of the PC activity (since its connection to DCN is inhibitory) which is specific and learned (through supervised learning at the granular cell - Purkinje cell synapses) from a general (nonspecific) activity term (from mossy fibers) which is approximately constant. Mind that, although the corrections of the DCN after learning may seem very irregular with high frequency terms (continuous lines Fig. 4(E) and (F)), the actual contributions are smoothed out by the motor system (in this case, the actual motor gears).

3.1. Robotic input/output

As briefly described in this work, cerebellar neural models are a current open issue whose operating basis is not yet well determined due to their working complexity principles. New tools for massive simulations (with multiple parameters) and state monitoring capabilities are necessary to identify how certain neural/subcircuit/neural layer features are related to the cerebellar functionality. Therefore, relating the cerebellar operation with the system in which the cerebellum is embodied seems to be the natural step forward. The presented integrated software platform is able to establish this interconnection between these two elements as shown in Figs. 4 and 5. Monitored snapshots of the whole cerebellar activity are related with their corresponding robot performance curve (system behavior). These snapshots facilitate the interpretation of the results giving a better insight about what is going on during embodied experimentation (behavioral experiments as the manipulation task illustrated in the previous section).

Fig. 5 is an illustrative example of the sort of performance curves that can be obtained by using the presented software platform.

Here, the robot arm is manipulating a 10 kg payload whilst executing a one second eight-like trajectory able to reveal the inner robot dynamics. Fig. 5(A) and (B), represent a snapshot of this one second eight-like trajectory execution in joint coordinates belonging to the initial learning stage (first row plots) and the final learning stage (position and velocity) (second row plots). The target trajectory at each joint is plotted in blue (continuous line) whilst the actual trajectory at each joint is plotted in red (dashed line). The error directionality (position and velocity error) is shown in these plots (either positive or negative error). As mentioned before, during the manipulation of objects with a significant weight, the arm–object platform dynamics differ from the original arm dynamics. This translates into a continuous negative error at the 2nd and 3rd joints which activates just one of the two inferior olive micro complexes (related to each joint) during the simulation. Additionally, Fig. 5(C) shows the Mean Absolute Error (MAE) obtained along the learning process. Finally, plot 5(D) represents just an example of how the obtained Cartesian coordinates of the tip of the robot arm evolve during the learning process. As shown, at the initial learning stage, the LWR is not capable of properly handling the attached payload; there is no acquired cerebellar corrective model for the 10 kg payload. Therefore, no corrective torque values are supplied yet. At the final learning stage, the cerebellum is able to provide the appropriate corrective torque terms achieving almost the aimed target trajectory.

Fig. 6 shows the same kind of experimentation conducted in Fig. 5 but extrapolated to different masses so as to reveal the capabilities and features that the learning at PF–PC synapses endows. Fig. 6(A) and (B), represent the MAE evolution whilst the robot arm is manipulating different payloads (10, 6 and 2.5 kg respectively) whether independently or consecutively. In Fig. 6(A) the learning process is reset, which means that all the synaptic weights at PF–PC are randomly chosen at the end of the learning of each payload whilst in Fig. 6(B) the learning process is not reset at the end of each payload learning. As can be seen, the learning is not destructive; the incoming learning process takes advantage of the previous learning process as indicated by the lower initial starting MAE error after switching between contexts. Fig. 6(C) points out the normalized performance that each of the aforementioned experiments achieves. Fig. 6(D) demonstrates how the learning process is compatible with incremental learning. Here, the payloads are switched every 50 trials (between 10 kg/6 kg in the left plot and 6 kg/2.5 kg in the right plot) thus showing how the learning process can simultaneously abstract two different payloads (two different dynamic models) that are only marginally interfering with each other.

3.2. Real time simulation

The computation load when simulating spiking neurons is high and needs to be done efficiently for controlling robots in real time. When any event-driven simulator is confronted with a massive amount of data to be processed online, this approach suffers due to the discontinuous flow of data to be computed. In fact, the learning process must be done online, in real time, as the robot is moving. A mechanism to ensure real time when processing all the neural activity involved during the simulation process has been implemented. During a neural simulation, all neural updates have to be processed in chronological order. However, during the neural simulation, future events may appear (i.e. events that occurred due to delayed spike firings or neural connections presenting delays). To manage this situation, a heap data structure able to efficiently insert and extract ordered events is required. Controlling the CPU time consumption of each time step allows real-time simulation. Although the calculation of the dynamics and kinematics of the robot (for instance, using a Newton Euler algorithm [74]) involves a constant number of operations at each time step, the neural simulation computational cost depends on the neural activity.

We have implemented a watchdog timer supervising each simulation time step. When the simulation process is consuming more time than a certain predefined constraint percentage of the total robot communication step time, the simulator skips non-critical event processing, thus keeping the simulation running in time (see Fig. 7). In our example, the total computation time has to remain below 2 ms, since the communication between the neural simulator and the robot platform is sliced in 2 ms intervals. As shown in Fig. 7(A), the computation time of each simulation slice (of 2 ms) consumes less than 2 ms. The ''computation time'' includes the cerebellar simulation time, the robotic simulation time, and the communication time between them. At each simulation step, the cerebellum updates and computes its internal neural states thus eliciting a set of generated spikes. There exists a close relationship between the number of generated spikes and the consumed computational time (Fig. 7(A) and (B)). In the end, a trade-off decision has to be taken. A watchdog ensures that the boundary will not be surpassed.

This illustrative simulation is composed by 1871 neurons and 69 603 synapses. We have used simple point neurons (parameterized according to different cerebellar neuron types) with three state variables (membrane potential and the excitatory and inhibitory conductances). Thus, 5613 state variables need to be continuously updated. During one second of simulation, the network produces 9890 spikes and 69 603 synaptic weight modifications (through spike time dependent plasticity at the parallel fiber to Purkinje cell synapses). All this needs to be computed within the real-time constraint. The simulation was run on a CPU consisting of a Pentium i7 3770k 3.4 GHz processor with 8 GB RAM all mounted on an ASUS P8Zseries motherboard.

4. Discussion

Along this paper, we have outlined how the EDLUT neural simulator has been equipped with an integrated robotic software framework. The dialog between these two elements, the EDLUT and the robotic software, is mediated by an efficient bidirectional interface (analog signals to spike patterns and vice versa) able to process sensory data from the robot agent and generate the appropriate robot motor commands. As a running embodied nervous-system example, we have implemented and described a cerebellar architecture within a robotic control closed-loop where the robot features allow the exploitation of the cerebellar potential in a manipulating control task. This manipulation task aims to follow a specific desired trajectory

consisting of sinusoidal components with the robotic arm manipulating a punctual mass. This punctual mass (representing the object under manipulation) affects the global dynamic model of the arm + object plant. The cerebellar system aims to provide corrective torque terms to compensate the existing mismatch between the arm dynamic model and the one of the arm + object under manipulation. These corrective torque terms are refined as the cerebellum acquires the dynamic model of the object under manipulation. This can be considered an abstraction process based on just the synaptic plasticity mechanism between the parallel fibers and the Purkinje cells.

The interest of this integrated neurobotics software platform can be outlined in two main points: for accelerating the development of biologically plausible control architectures cooperating with robot agents and for studying how certain capabilities of the cerebellum in coordinated motion and object manipulation are based on cellular characteristics, nervous system topology, or local synaptic adaptation mechanisms. In fact, a rich dynamical environment (i.e. highly reconfigurable robot model dynamics and reconfigurable cerebellar control loops) is a powerful tool to explore neurophysiological hypotheses from a functional point of view. All this also needs to be complemented with an appropriate monitoring and evaluation methodology. Here, it has been addressed not only just the way in which the neural activity can be plotted and interpreted by considering the micro-complex biologically plausible cerebellar organization, but also the neural activity contributions to agonist and antagonist motor system outputs thanks to the continuous monitoring of the target and actual joint trajectories.

Furthermore, the performance obtained is also remarkable. Although a simulation achieving real-time could be considered to be irrelevant, it is a critical non-trivial issue in embodied system neuroscience. When doing experiments with a real neuro-operated body, real-time operation becomes a major requirement. We have shown how this integrated software framework fulfils real-time requirement enabling a future real-robot cerebellar spiking control. In fact, the software framework integrating the neural simulator, the robotic simulator and all the communication and monitoring components has been developed with demanding real-time constraints.

5. Conclusions

In this paper, we show how a cerebellar structure integrated in the control loop as an adaptive feedforward model can learn to abstract model dynamics of objects being manipulated. We use an integrated simulation platform consisting of a *real-time* spiking neural simulator (EDLUT) and a simulated robot (LWR). This platform allows us to monitor the cell activity at different layers in terms of spike patterns as well as the contribution that they produce in terms of actual corrective torques within the control loop before learning the object model, and also eventually in the corrected trajectory (closer to the goal trajectory) after the learning process converges. The possibility of monitoring each cell activity allows us to interpret how the whole network works, receiving distributed spike patterns from the mossy fibers, producing sparse coding at the granular layer and adapting the weights between the granular layer and the Purkinje cells through supervised learning driven by the inferior olive activity (which is related to the actual error at each instant of the trajectory execution). The cerebellum integrated in the control loop with the presented configuration (actual and desired positions/velocities reaching the cerebellum through mossy fibers), performs the model abstraction process, as a function approximation problem (with the object under manipulation on-the-loop).

In the final experiments done (Fig. 6(A), (B) and (C)), we demonstrate that the presented architecture can learn dynamic models incrementally (with low interference with each other). In fact, learning a new model takes advantage of previous learned weights (related to previous objects under manipulation) but without destroying these previous models (Fig. 6(D)).

Acknowledgments

This work was supported by the EU grants REALNET FP7-ICT270434 (where the cerebellar simulations were developed) and HBP FP7 Flagship Project 604102 (monitoring tools are being developed), and by the national grants ARC-VISION (TEC2010-15396) and PYR (2014-16).

Appendix A. Considerations related to benchmark trajectory accuracy; communication interface delay

When a real robot is connected to the controller (cerebellar base controller), its communication interface introduces a delay each time that the joint positions are obtained and the joint motor torques are set. This delay limits the frequency in which the controller can interact with the robot. Thus, the robot communication interface determines the minimum control cycle time. The robot trajectory accuracy decreases as the control cycle time increases, since, for example, the robot motor torque set points remain constant during each cycle. Therefore, the suitability of a concrete communication interface (bus) depends on the trajectory accuracy decline which is acceptable. It is of importance then to take this limitation into account when developing realistic real-time software towards embodied system neuroscience. Spiking cerebellar updating usually demands simulation step times in the millisecond scale (1–2 ms) [5–7] making this bus delay consideration an important factor to be considered when designing cerebellar control stages.

Just as an example, Fig. A.1 illustrates the inaccuracy introduced by different bus transmission delays for different conducted experiments using a simulated lightweight robot [77] and an eight-shaped test trajectory. In order to simulate the effect of a communication bus, the torque generated by the controller is repeatedly kept constant for a period (control cycle time). When the robot input torque is increasing, the bus delay produces an average torque below the desired one (with negligible bus delay). The opposite occurs when the input torque is decreasing. Therefore, the joint angle error caused by the transmission time is related to the desired angle value and velocity during the trajectory execution.

Appendix B. Considerations related to the friction force of the robot joints

There are several forces that affect the expected robot dynamic model. When these forces are not properly taken into account, an open-loop controller for an ideal robot may fail to produce accurate movements. The most relevant perturbing forces that can be easily found in simple robotic arms can be summarized as follows:

Force exerted by the wires attached to the robot motors (for supping current and measuring angle encoder inputs/outputs): These forces remain relatively low. They can pull or push the robot's joints when the arm is in certain positions, facilitating or hindering the movement in certain directions. Since these forces are usually very low, it can be assumed that they will be compensated thanks to the adaptability of the cerebellar controller.

Inner dry friction forces of the robot joints: The two regimes of dry friction are static friction (the joint remains static) and kinetic friction (between moving surfaces of the joint). Sometimes the static friction of some robots is very significant. This friction force can be also compensated by the cerebellar controller. Nevertheless, when the magnitude of this force is comparable or higher than the rest of the force that the cerebellar controller must exert (to compensate for other deviations from the ideal dynamic model), the precision of the adaptive cerebellar module to compensate these other deviations is low. This occurs because if the cerebellar output force range increases, the resolution of its output per force unit decreases. This output range increase is equivalent to multiplying the output by a factor; therefore, the inaccuracy of this output would also be multiplied.

Accurately compensating the effect of the friction forces can sometimes become considerably complex, depending on the used compensation technique (this force is not the same in all the possible joint angles); in fact, the friction term proves to be crucial when controlling light-weight robot arms with high-ratio gear boxes because there are no standard methodologies/techniques to control these robots without massive modeling [76]. However a complex technique to fully compensate this force is not needed since the cerebellar module can conveniently compensate it (when it is relatively low). Thus, in this case, the goal of the compensation technique should not be to fully compensate for these perturbations, but to keep them in a range domain where the cerebellar module can learn to accurately correct the movement deviations.

References

[1] J.S. Albus, A theory of cerebellar function, Math. Biosci. 10 (1971) 25–61.
[2] D. Marr, A theory of cerebellar cortex, J. Physiol. 202 (1969) 437–470.
[3] S.F. Giszter, K.A. Moxon, I.A. Rybak, J.K. Chapin, Neurobiological and neurorobotic approaches to control architectures for a humanoid motor system, Robot. Auton. Syst. 37 (2001) 219–235.
[4] E. Ros, R. Carrillo, E.M. Ortigosa, B. Barbour, R. Agís, Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics, Neural Comput. 18 (2006) 2959–2993.
[5] N.R. Luque, J.A. Garrido, R.R. Carrillo, S. Tolu, E. Ros, Adaptive cerebellar spiking model embedded in the control loop: context switching and robustness against noise, Int. J. Neural Syst. 21 (2011) 385–401.
[6] N.R. Luque, J.A. Garrido, R.R. Carrillo, O.J.M.D. Coenen, E. Ros, Cerebellarlike corrective model inference engine for manipulation tasks, IEEE Trans. Syst. Man Cybern. B 41 (2011) 1299–1312.
[7] N.R. Luque, J.A. Garrido, R.R. Carrillo, O.J.M.D. Coenen, E. Ros, Cerebellar input configuration toward object model abstraction in manipulation tasks, IEEE Trans. Neural Netw. 22 (2011) 1321–1328.
[8] J.S. Albus, Data storage in the cerebellar model articulation controller (CMAC), Trans. ASME, J. Dyn. Syst. Meas. Control 3 (1975) 228–233.
[9] C. Sabourin, O. Bruneau, Robustness of the dynamic walk of a biped robot subjected to disturbing external forces by using CMAC neural networks, Robot. Auton. Syst. 51 (2005) 81–99.
[10] C.K. Tham, Reinforcement learning of multiple tasks using a hierarchical CMAC architecture, Robot. Auton. Syst. 15 (1995) 247–274.
[11] T. Yamazaki, S. Nagao, A computational mechanism for unified gain and timing control in the cerebellum, PLoS One 3 (2012) e33319.
[12] T. Yamazaki, S. Tanaka, Neural modeling of an internal clock, Neural Comput. 17 (2005) 1032–1058.
[13] T. Yamazaki, S. Tanaka, The cerebellum as a liquid state machine, Neural Netw. 20 (2007) 290–297.
[14] T. Yamazaki, S. Tanaka, Computational models of timing mechanisms in the cerebellar

granular layer, Cerebellum 8 (2009) 423–432.
[15] T. Yamazaki, S. Tanaka, A spiking network model for passage-of-time representation in the cerebellum, Eur. J. Neurosci. 26 (2007) 2279–2292.
[16] P. Manoonpong, T. Geng, T. Kulvicius, B. Porr, F. Wörgötter, Adaptive, fast walking in a biped robot under neuronal control and learning, PLoS Comput. Biol. 3 (2007) e134.
[17] W. Wolpert, M. Kawato, Multiple paired forward and inverse models for motor control, Neural Netw. 11 (1998) 1317–1329.
[18] P. Dean, J. Porril, Adaptive filter models of the cerebellum. computational analysis, Cerebellum 7 (2008) 567–571.
[19] P. Dean, J. Porrill, The cerebellum as an adaptive filter: a general model? Funct. Neurol. 25 (2010) 173–180.
[20] P. Dean, J. Porrill, C.F. Ekerot, H. Jörntel, The cerebellar microcircuit as an adaptive filter: experimental and computational evidence, Nat. Rev. Neurosci. 11 (2010) 30–34.
[21] M. Fujita, Adaptive filter model of the cerebellum, Biol. Cybernet. 45 (1982) 195–206.
[22] J. Porrill, P. Dean, Cerebellar motor learning: when is cortical plasticity not enough? PLoS Comput. Biol. 3 (2007) e197.
[23] J.C. Houk, J.T. Buckingham, A.G. Barto, Models of cerebellum and motor learning, Behav. Brain Sci. 19 (1996) 369–383.
[24] S. Tolu, M. Vanegas, N.R. Luque, J.A. Garrido, E. Ros, Bio-inspired adaptive feedback error learning architecture for motor control, Biol. Cybernet. 106 (2012) 507–522.
[25] S. Tolu, M. Vanegas, J.A. Garrido, N.R. Luque, E. Ros, Adaptive and predictive control of a simulated robot arm, Int. J. Neural Syst. 23 (2013).
[26] N. Schweighofer, J. Spoelstra, M.A. Arbib, M. Kawato, Role of the cerebellum in reaching movements in human. II. A neural model of the intermediate cerebellum, Eur. J. Neurosci. 10 (1998) 95–105.
[27] N. Schweighofer, M.A. Arbib, M. Kawato, Role of the cerebellum in reaching movements in human. I. Distributed Inverse dynamics control, Eur. J. Neurosci. 10 (1998) 86–94.
[28] J.C. Eccles, Circuits in the cerebellar control of movement, Proc. Natl. Acad. Sci. 58 (1967) 336–343.
[29] M. Ito, Adaptive modification of the vestibulo-ocular reflex in rabbits affected by visual inputs and its possible neuronal mechanisms, in: R. Granit, O. Pompeiano (Eds.), Progress in Brain Research, Elsevier, 1979, pp. 757–761.
[30] M. Ito, The Cerebellum and Neural Control, Raven Press, New York, 1984.
[31] M. Ito, Synaptic plasticity in the cerebellar cortex and its role in motor learning, Can. J. Neurol. Sci.. Le J. Can. Sci. Neurol. 20 (Suppl 3) (1993) S70–S74.
[32] M. Ito, Control of mental activities by internal models in the cerebellum, Nat. Rev. Neurosci. 9 (2008) 304–313.
[33] E. D'Angelo, C.I. De Zeeuw, Timing and plasticity in the cerebellum: focus on the granular layer, Trends Neurosci. 32 (2009) 10.
[34] Z. Gao, B.J. vanBeugen, C.I. De Zeeuw, Distributed synergistic plasticity and cerebellar learning, Nat. Rev. Neurosci. 13 (2012) 1–17.
[35] C. Hansel, D.J. Linden, E. D'Angelo, Beyond parallel fiber LTD: the diversity of synaptic and non-synaptic plasticity in the cerebellum, Nat. Neurosci. 4 (2001) 467–475.
[36] G.A. Jacobson, D. Rokni, Y. Yarom, A model of the olivo-cerebellar system as a temporal pattern generator, Trends Neurosci. 31 (2008) 617–625.
[37] G.A. Jacobson, I. Lev, Y. Yarom, D. Cohen, Invariant phase structure of olivo- cerebellar oscillations and its putative role in temporal pattern generation, Proc. Natl. Acad. Sci. 106 (2009) 3579–3584.
[38] R.R. Carrillo, E. Ros, S. Tolu, T. Nieus, E. D'Angelo, Event-driven simulation of cerebellar granule cells, Biosystems 94 (2008) 10–17.
[39] S. Solinas, T. Nieus, E. D'Angelo, A realistic large-scale model of the cerebellum granular layer predicts circuit spatio temporal filtering properties, Front. Cell. Neurosci. 4 (2010).
[40] P. Gleeson, V. Steuber, R.A. Silver, S. Crook, NeuroML, in: Computational Systems Neurobiology, Springer, 2012, pp. 489–517.
[41] M.L. Hines, T. Morse, M. Migliore, N.T. Carnevale, G.M. Shepherd, ModelDB: a database to support computational neuroscience, J. Comput. Neurosci. 17 (2004) 7–11.
[42] N.C. Rowland, D. Jaeger, Coding of tactile response properties in the rat deep cerebellar

nuclei, J. Neurophysiol. 94 (2005) 1236–1251.
[43] N.C. Rowland, D. Jaeger, Responses to tactile stimulation in deep cerebellar nucleus neurons result from recurrent activation in multiple pathways, J. Neurophysiol. 99 (2008) 704–717.
[44] T.M. Teune, J. van der Burg, C.I. de Zeeuw, J. Voogd, T.J. Ruigrok, Single Purkinje cell can innervate multiple classes of projection neurons in the cerebellar nuclei of the rat: a light microscopic and ultrastructural triple-tracer study in the rat, J. Comp. Neurol. 392 (1998) 164–178.
[45] W. Zhang, W.D. Linden, Long-term depression at the mossy fiber-deep cerebellar nucleus synapse, J. Neurosci. 26 (2006) 6935–6944.
[46] A. Delorme, J. Gautrais, R. van Rullen, S. Thorpe, SpikeNET: a simulator for modeling large networks of integrate and fire neurons, Neurocomputing 26 (1999) 989–996.
[47] L. Watts, Event-driven simulation of networks of spiking neurons, Adv. Neural Inf. Process. Syst. (1994) 927–934.
[48] M. D'Haene, B. Schrauwen, J. Van Campenhout, D. Stroobandt, Accelerating event-driven simulation of spiking neurons with multiple synaptic time constants, Neural Comput. 21 (2009) 1068–1099.
[49] T. Makino, A discrete-event neural network simulator for general neuron models, Neural Comput. Appl. 11 (2003) 210–223.
[50] W. Gerstner, W.M. Kistler, Spiking Neuron Models: Single Neurons, Popula- tions, Plasticity, Cambridge University Press, 2002.
[51] J.A. Garrido, in: Edlut official website, 2012.
[52] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J.M. Bower, M. Diesmann, A. Morrison, P.H. Goodman, Simulation of networks of spiking neurons: a review of tools and strategies, J. Comput. Neurosci. 23 (2007) 349–398.
[53] J. Reutimann, M. Giugliano, S. Fusi, Event-driven simulation of spiking neurons with stochastic dynamics, Neural Comput. 15 (2003) 811–830.
[54] J.A. Garrido, R.R. Carrillo, N.R. Luque, E. Ros, Event and time driven hybrid simulation of spiking neural networks, in: Advances in Computational Intelligence, Springer, 2011, pp. 554–561.
[55] A. Pouget, P. Dayan, R. Zemel, Information processing with population codes, Nat. Rev. Neurosci. 1 (2000) 125–132.
[56] S. Wu, S. Amari, H. Nakahara, Population coding and decoding in a neural field: a computational study, Neural Comput. 14 (2002) 999–1026.
[57] T. Flash, T.J. Sejnowski, Computational approaches to motor control, Curr. Opin. Neurobiol. 11 (2001) 655–662.
[58] B. Amirikian, A.P. Georgopoulos, Modular organization of directionally tuned cells in the motor cortex: is there a short-range order? Proc. Natl. Acad. Sci. 100 (2003) 12474–12479.
[59] K.R. Boff, J.E. Lincoln, Engineering Data Compendium. Human Perception and Performance. Vol. 3, Harry G Armstrong. Aerospace Medical Research Lab Wright-Patterson Afb Oh, 1988.
[60] N.V. Swindale, Orientation tuning curves: empirical description and estima- tion of parameters, Biol. Cybernet. 78 (1998) 45–56.
[61] N.R. Luque, J.A. Garrido, J. Ralli, J.J. Laredo, E. Ros, From sensors to spikes: evolving receptive fields to enhance sensorimotor information in a robot-arm, Int. J. Neural Syst. 22 (2012) 1250013.
[62] J.D. Victor, Spike train metrics, Curr. Opin. Neurobiol. 15 (2005) 585–592.
[63] M.C. van Rossum, A novel spike distance, Neural Comput. 13 (2001) 751–763.
[64] B. Schrauwen, J. Van Campenhout, BSA, a fast and accurate spike train encoding scheme, in: Proceedings of the International Joint Conference on Neural Networks, 2003, IEEE, 2003, pp. 2825–2830.
[65] G.C. Goodwin, Adaptive Prediction and Control, Prentice Hall, NJ, 1984.
[66] R.C. Miall, D.J. Weir, D.M. Wolpert, J.F. Stein, Is the cerebellum a Smith predictor? J. Mot. Behav. 25 (1993) 203–216.
[67] D.M. Wolpert, R.C. Miall, Forward models for physiological motor control, Neural Netw. 9 (1996) 1265–1279.
[68] A.J. Bastian, Learning to predict the future: the cerebellum adapts feedforward movement control, Curr. Opin. Neurobiol. 16 (2006) 645–649.

[69] E.J. Hwang, R. Shadmehr, Internal models of limb dynamic and the encoding of limb state, J. Neural Eng. 2 (2005) 266–278.
[70] E. Nakano, H. Imamizu, R. Osu, Y. Uno, H. Gomi, T. Yoshioka, M. Kawato, Quantitative examinations of internal representations for arm trajectory planning. Minimum commanded torque change model, J. Neurophysiol. 81 (1999) 2140–2155.
[71] E. Todorov, Optimality principles in sensorimotor control (review), Nat. Neurosci. 7 (2004) 907–915.
[72] E.R. Kandel, J.H. Schwartz, T.M. Jessell, Principles of Neural Science, McGraw- Hill Professional Publishing, New York, 2000.
[73] M. Kawato, K. Furukawa, R. Suzuki, A hierarchical neural-network model for control and learning of voluntary movement, Biol. Cybernet. 57 (1987) 169–185.
[74] J.A. Garrido, N.R. Luque, E. D'Angelo, E. Ros, Distributed cerebellar plasticity implements adaptable gain control in a manipulation task: a closed-loop robotic simulation, Front. Neural Circuits 7 (2013).
[75] B. Siciliano, O. Khatib, Springer Handbook of Robotics, Springer, 2008.
[76] P. van der Smagt, Benchmarking cerebellar control, Robot. Auton. Syst. 32 (2000) 237–251.
[77] A. Albu-Schäffer, S. Haddadin, C. Ott, A. Stemmer, T. Wimböck, G. Hirzinger, The DLR lightweight robot: design and control concepts for robots in human environments, Int. J. Ind. Robot 34 (2007) 376–385.
[78] G. Hirzinger, J. Butterfab, M. Fischer, M. Grebenstein, M. Hähnle, H. Liu, N. Shäfer, I. Sporer, A mechatronics approach to the design of light-weight arms and multifingered hands, in: ICRA, 2000, pp. 46–54.
[79] R.E. Kettner, S. Mahamud, H. Leung, N. Sittko, J.C. Houk, B.W. Peterson, A.G. Barto, Prediction of complex two-dimensional trajectories by a cerebellar model of smooth pursuit eye movement, J. Neurophysiol. 77 (1997) 2115–2130.
[80] H. Hoffmann, G. Petkos, S. Bitzer, S. Vijayakumar, Sensor-assisted adaptive motor control under continuously varying context, 2007.
[81] N. Schweighofer, K. Doya, M. Kawato, Electrophysiological properties of inferior olive neurons: a compartmental model, J. Neurophysiol. 82 (1999) 804–817.

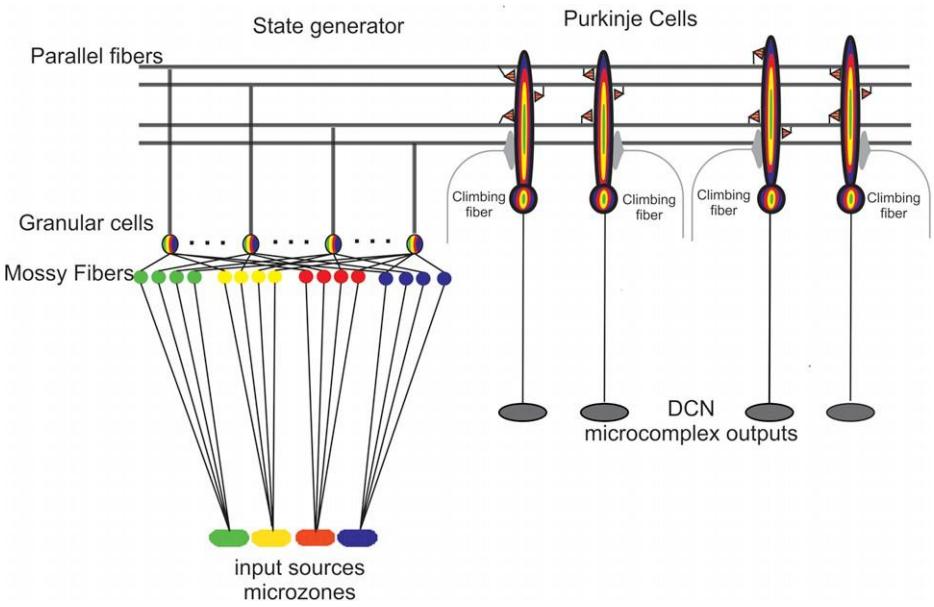

Fig. 1. Cerebellar architecture. Color representation indicates signals from different sources such as different cuneate receptive fields or proprioceptors. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

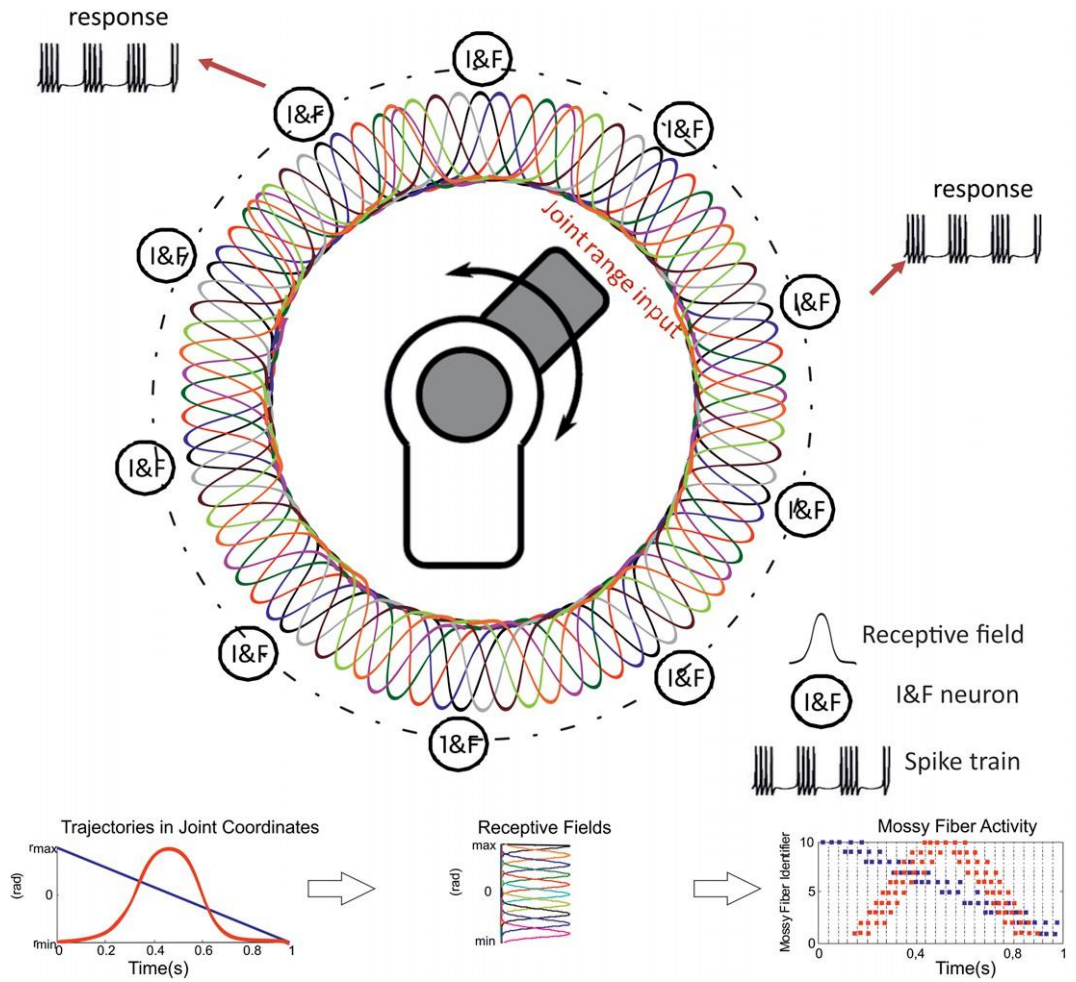

Fig. 2. Population coding of input (proprioceptors) signals. The joint angle position (input signal) provided by a joint encoder which covers the joint range is translated into a population coding whilst a certain trajectory is followed using a set of tuning receptive fields (Gaussian like curves) which represent the current injected into spiking neurons by different sensory receptors (proprioceptors). Each proprioceptor's value (output signal) is integrated using an integrate-and-fire neuron model and determines the activity response of an input neuron (as is the case of Mossy Fiber neurons belonging to the cerebellar circuitry). Lower plots illustrate how two trajectories (encoder angle varying in time) defined in a single joint produce spiking patterns when the contributions to the integrate neurons are integrated through the sensory receptive fields.

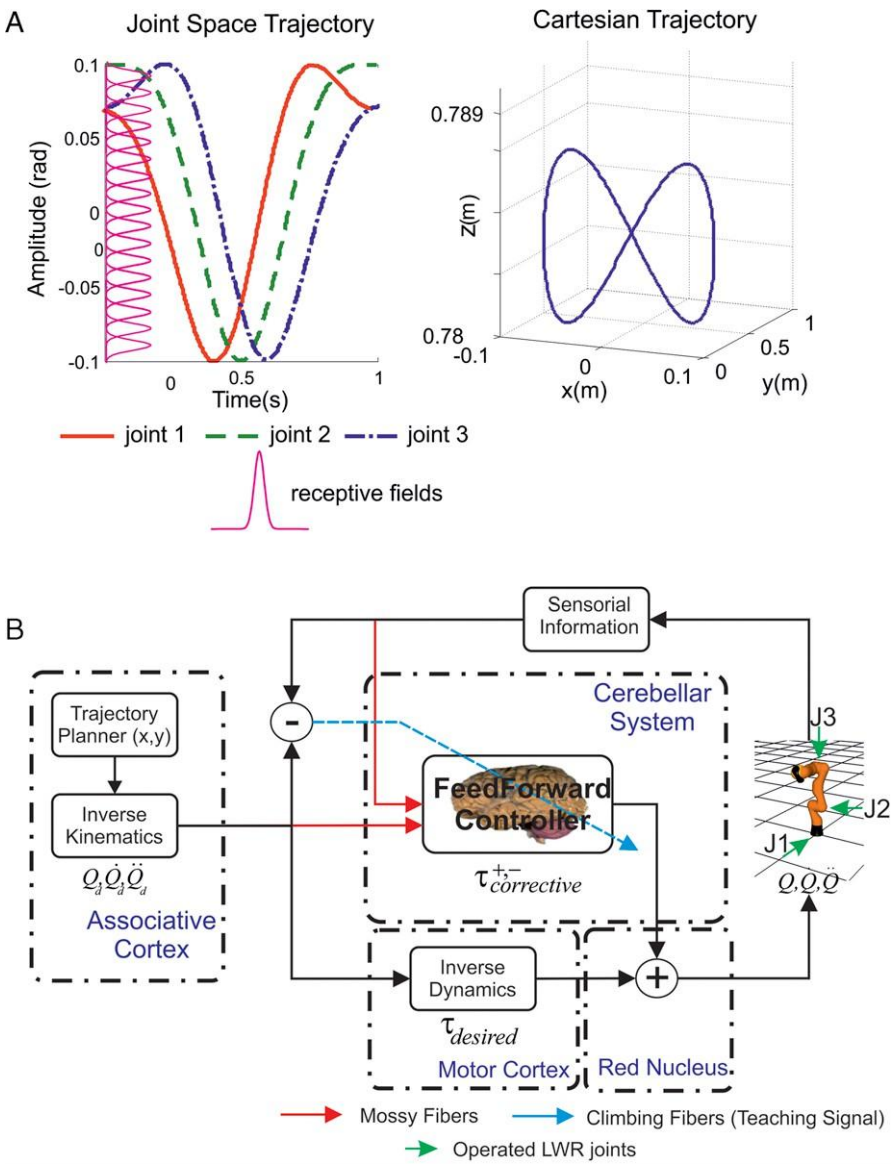

Fig. 3. (A) Benchmark trajectory to be performed consisting of sinusoidal components. The trajectory is shown in both joint coordinate and Cartesian coordinates (eight-like trajectory). The receptive fields are distributed covering the whole range determined by the joint coordinates. (B) Implemented cerebellar control loop. The cerebellum infers a corrective model that produces effective corrective commands in order to compensate the existing mismatch between the crude inverse dynamic robot model and the actual base dynamic plant model. The desired arm states are generated according to the Cartesian trajectory to be followed (positions (Q_d), velocities ($\dot{Q}_d$) and accelerations ($\ddot{Q}_d$)) by the trajectory generator (a crude inverse kinematic model representing the output of the associative cortex and other motor areas). These desired arm states in joint coordinates are used at each time step to compute desired torque commands (crude inverse dynamic robot model). They are also used as input to the cerebellum which produces the predictive corrective commands ($\tau_{corrective}$) which are added to these crude torque commands ($\tau_{desired}$). The final total torque addition is supplied to the robot plant. The difference between the actual robot trajectory and the desired one is used to calculate the climbing fiber activity which is supplied to the cerebellum as a teaching input signal (for adapting PF–PC synaptic weights).

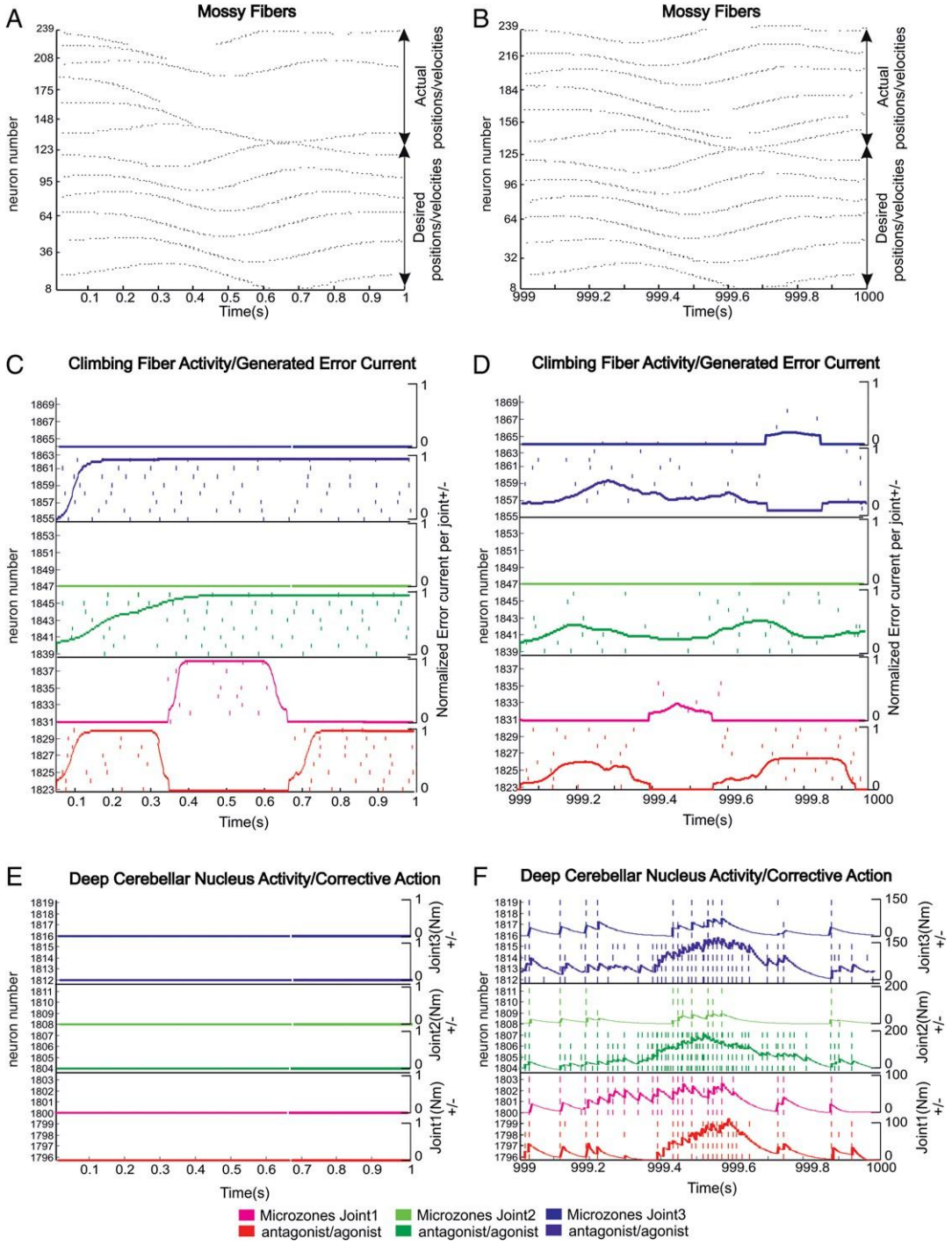

Fig. 4. Cerebellar activity monitoring one second simulation snapshot at the beginning of the learning process (left plots) and at the end of the learning process (right plots). Left *Y* axes are used for the neuron number in the network. The bottom legend indicates how these neurons are related to different joints and agonist or antagonist micro-complexes by using different colors. Plots C–F include two overlapped representations, the spike patterns related to the left *Y* axis and a continuous line referred to the right *Y* axis at each plot. (A) (B) Translation of the desired/actual joint positions/velocities into mossy fiber activity at the beginning of the learning process (A) and at the final learning stage (B). (C) (D) Evolution of the climbing fiber activity during the learning process and its corresponding error current proportional to the actual position and velocity error. (C) High error current translated into spikes at the initial learning stage. (D) Lower error current translated into spikes at the end of the learning process. (E) (F) Cerebellar output during the learning process and the corresponding generated analog corrective action. (E) Cerebellar output at the beginning of the learning process. No spikes are elicited at the DCNs, the corrective actions are

zero. (F) Cerebellar output at the end of the learning process. The spike output activity is translated into corrective actions for each robot joint. Each couple of micro-complexes is related to a certain robot joint (agonist and antagonist terms). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

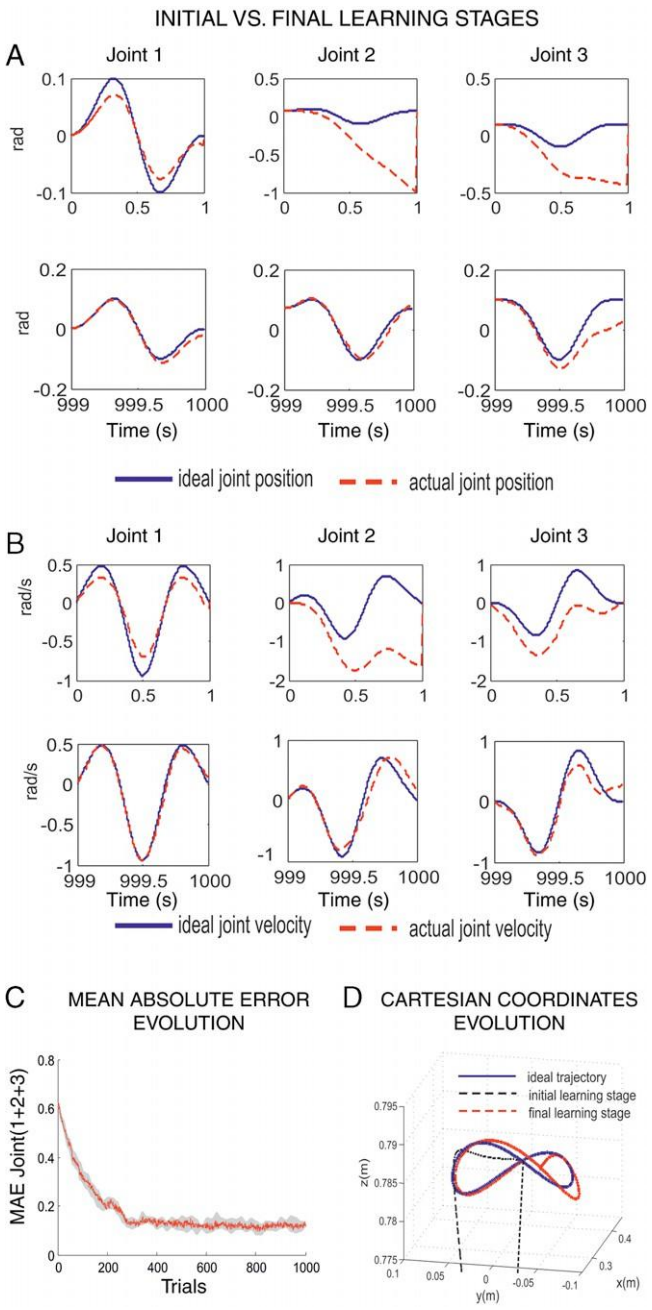

Fig. 5. Robotic performance (system behavior) in a manipulation task. The manipulation of a 10 kg payload whilst executing an eight-like trajectory reveals the inner robot dynamics. The benchmark trajectory execution takes one second in each trial. (A) Snapshot of the execution of the eight-like trajectory in joint coordinates (position) belonging to the initial learning stage (top plots) and the final learning stage (bottom plots). (B) Snapshot of the execution of the eight-like velocity trajectory in joint coordinates (velocity) belonging to the initial learning stage (top plots) and the final learning stage (bottom plots). (C) Averaged Mean Absolute Error (during each trial) obtained along the learning process computing the addition of the individual MAEs corresponding to each robot joint. Four different simulations with different initial random values at PF–PC synaptic weights have been used. The shadowed area is defined between the maximum and minimum values among the four simulations in each trial. The red curve is the average of the four

simulations. (D) Cartesian coordinate evolution during the learning process. At the initial learning stage, the LWR is not capable of properly handling the attached payload. At the final learning stage, the cerebellum is able to provide the appropriate corrective torque values achieving almost the aimed target trajectory. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

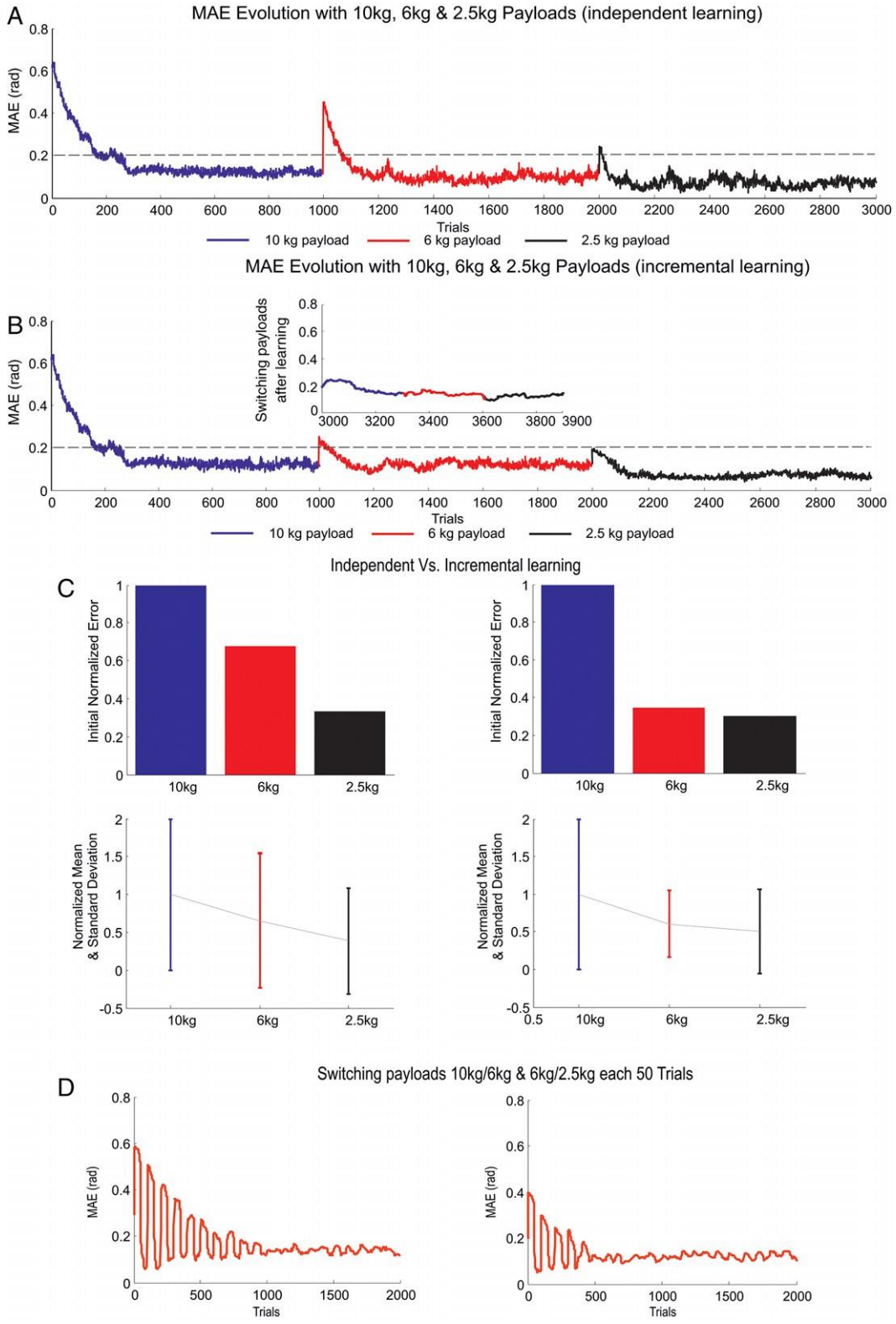

Fig. 6. Independent Learning vs. Incremental Learning. (A) Manipulation of 10, 6, and 2.5 kg independently. The learning process is reset (synaptic weights at PF–PC are randomly chosen at

the end of the learning process of each payload). (B) Manipulation of 10, 6, and 2.5 kg consecutively. The learning process is not reset at the end of the each payload learning. The learning is not destructive; the incoming learning process takes advantage of the previous learning process as indicated by the lower initial starting MAE error after switching between contexts (objects under manipulation). The zoom in the graph shows how the system behaves when these new objects are presented again (300 iterations each). This demonstrates that the learning is done with only low interference between the object model dynamics being learned (abstracted). (C) Normalized initial error values (obtained in the ten-first trial errors per payload, 10 kg initial error has been taken as the worse possible scenario) obtained at the beginning of the learning process with independent learning (left plots) and consecutive or incremental learning (right plots). The normalized average and standard deviation of MAE values (of the last 100 trials of each learning process) with independent learning (left plot) and consecutive or incremental learning (right plot) are also shown. In any case, incremental learning outperforms independent learning. (D) Incremental learning. Switching payloads every 50 trials (between 10 kg/6 kg in the left plot and 6 kg/2.5 kg in the right plot). It is shown how the learning can simultaneously abstract two different payloads (two different dynamic models) only marginally interfering with each other.

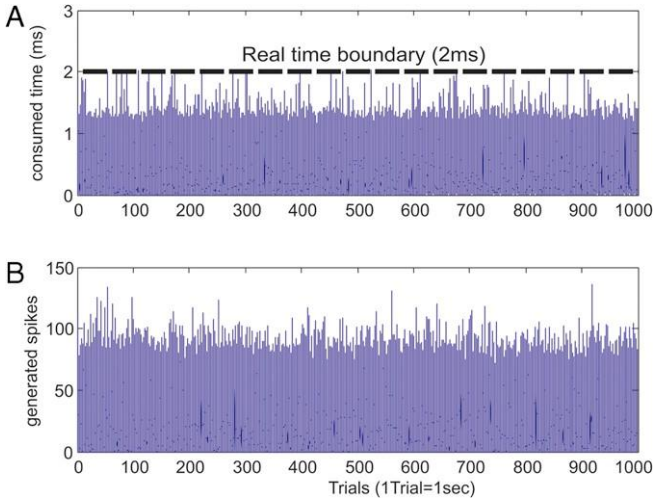

Fig. 7. Real time monitoring. The total computation time has to remain below 2 ms, because the communication between the neural simulator and the robot platform (real or simulated) is sliced in intervals of 2 ms.

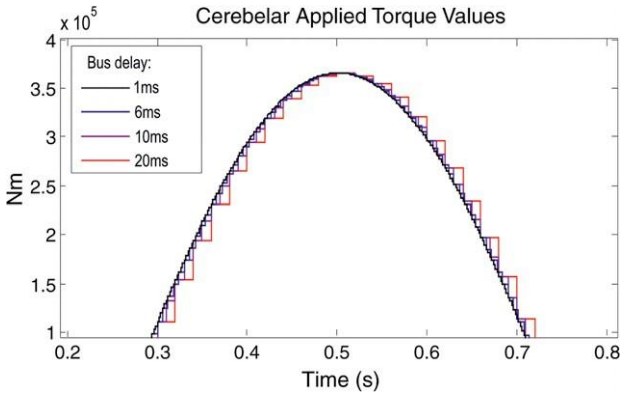

Fig. A.1. Possible consequences of the interface delay: snapshot of the cerebellar torque supplied to a LWR robot [77] (after being kept constant for several milliseconds as indicated in different traces).