Liquid Computing in a Simplified Model of Cortical Layer IV: Learning to Balance a Ball

Dimitri Probst[1,3], Wolfgang Maass[2], Henry Markram[1], and Marc-Oliver Gewaltig[1]

[1] Blue Brain Project, École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland
marc-oliver.gewaltig@epfl.ch
[2] Institute for Theoretical Computer Science, Technische Universität Graz, A-8010 Graz, Austria
[3] Ruprecht-Karls Universität Heidelberg, D-69117 Heidelberg, Germany

Abstract. We present a biologically inspired recurrent network of spiking neurons and a learning rule that enables the network to balance a ball on a flat circular arena and to steer it towards a target field, by controlling the inclination angles of the arena. The neural controller is a recurrent network of adaptive exponential integrate and fire neurons configured and connected to match properties of cortical layer IV. The network is used as a liquid state machine with four action cells as readout neurons. The solution of the task requires the controller to take its own reaction time into account by anticipating the future state of the controlled system. We demonstrate that the cortical network can robustly learn this task using a supervised learning rule that penalizes the error on the force applied to the arena.

Keywords: Spiking neural networks, NEST, dynamic control task, neurobotics, brain-inspired computing, AdEx, supervised learning, closed-loop motor control.

1 Introduction

Controlling a physical dynamical system with a realistic spiking neuronal circuit in a closed loop is difficult for several reasons. First, the intrinsic time constants of the physical system and those of the neurons may differ greatly. Second, the neural controller must implicitly learn the kinematics of the dynamical system. And third, in closed perception-action loops, small errors of the controller may amplify to the point where the system can no longer be controlled. Joshi and Maass [1] used a liquid-state machine [4] based on a general microcircuit model and linear readout units to control the movements of a robot arm. In an alternative approach, Bouganis and Shanahan [3] used spike-timing dependent plasticity (STDP) to teach a spiking network how to control a robot arm with four degrees of freedom. In this work, we trained a cortex-like network to solve a dynamic control task.

A.E.P. Villa et al. (Eds.): ICANN 2012, Part I, LNCS 7552, pp. 209–216, 2012.

Dynamic control tasks, like pole-balancing or balancing a ball on a moving surface, are particularly challenging because the system continues to evolve while the controller determines its next action and by the time the controller applies its force, it may no longer be appropriate. Thus, such tasks can only be solved if the controller takes its own reaction time into account by anticipating the state of the system when the control force is actually applied.

In this paper, we investigate if a recurrent network of spiking neurons, mimicking a piece of layer IV of a rat somatosensory cortex, can be trained to steer a rolling ball towards a target by controlling the inclination angles of the surface that the ball is rolling on. Following earlier work of Maass et al. [4], we use liquid computing to solve this task. In this paradigm, the recurrent network is used to map a spatio-temporal stimulus into a high-dimensional space. The desired response or target function is then constructed by readout neurons or action cells which compute a weighted sum of the high-dimensional activity in the recurrent network. In our case, there are four action cells, controlling the forces which can change the inclination angles of the arena.

2 Control Task and Network Architecture

The system to be controlled consists of a ball with mass m_{ball} and radius r_{ball} that rolls on a flat circular arena with a reflecting border at radius r_{arena} (Figure 1). The inclination of the arena is determined by two orthogonal angles α and β. The ball can roll on the arena and is subject to two forces, gravity and rolling resistance. If the arena is tilted against the horizontal plane, gravity will

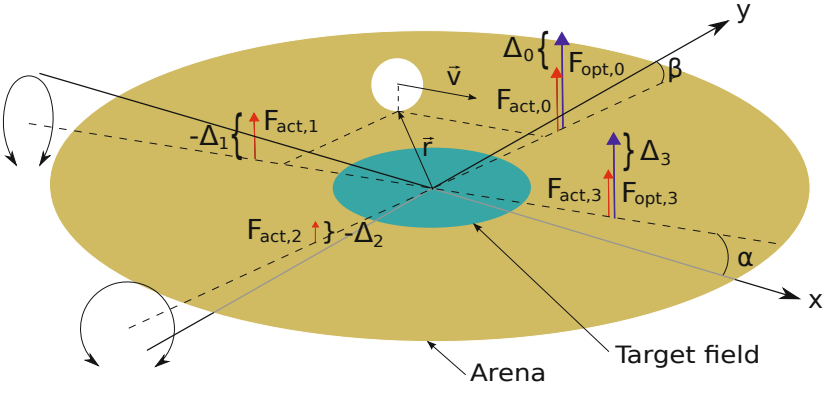

Fig. 1. Illustration of the control task: A ball (*white*) of mass m_{ball} moves on a circular arena (*yellow*) which can be rotated around two orthogonal axes. The goal of the task is to bring the ball into the target field (*blue*) by tilting the arena. In order to change the inclination angles, the forces $F_{\text{act},0}$ to $F_{\text{act},3}$ are applied which compose $\boldsymbol{F}_{\mathbf{act}}$. Depending on the current position $\boldsymbol{r}$ and velocity $\boldsymbol{v}$, an optimal force $\boldsymbol{F}_{\mathbf{opt}}$ is computed according to Equation 3. Here, the components $F_{\text{opt},1}$ and $F_{\text{opt},2}$ are zero. The differences Δ_0 to Δ_3 are used for learning. Note, sizes are not to scale.

accelerate the ball, thereby increasing its kinetic energy. If the arena is aligned with the horizontal plane, rolling resistance will be maximal and the kinetic energy will dissipate until the ball comes to rest. The strength of this dissipative force is determined by the coefficient of rolling friction η and the coefficient of restitution ρ.

The controller should solve the following task. Starting from an arbitrary initial position and velocity (within appropriate ranges) and given the current position of the ball, the controller must apply the appropriate forces to steer the ball into the target field at the center of the arena.

The forces are applied at four force points $F_{\text{act},0}$, $F_{\text{act},1}$, $F_{\text{act},2}$, and $F_{\text{act},3}$ (cf. Figure 1). Each angle is controlled by two opposing forces, similar to antagonistic muscle pairs.

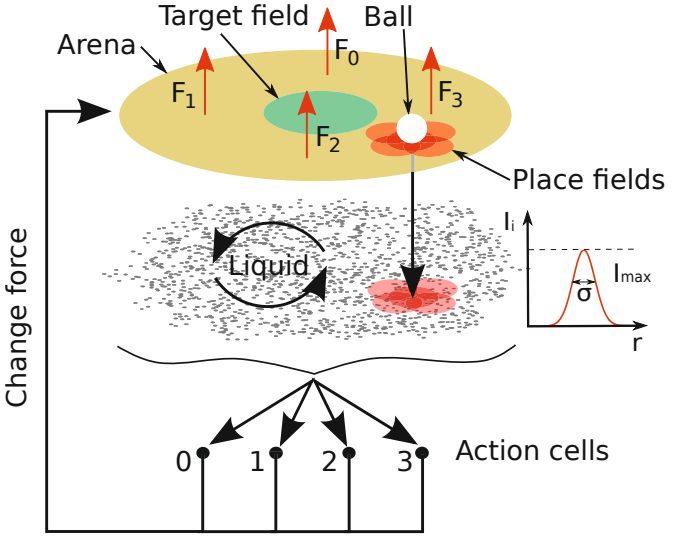

Fig. 2. Illustration of the closed-loop motor control: Neurons in the liquid (*grey dots*) receive position-dependent Gaussian activation currents from within their *place fields* (*red*). The neurons are randomly distributed in a hexagon which is topologically mapped onto the arena. The resulting spiking activity reverberates in the liquid through recurrent connections. Four action-cells integrate a weighted sum of the activity in the liquid. Finally, the membrane potential of the action cells is transformed to a motor force which changes the inclination of the arena. Drawing not to scale.

As neural controller we use a recurrent network of adaptive exponential integrate and fire neurons [5] and four non-spiking action cells, each representing a population of motor neurons. The properties of the neurons and their connections are chosen to mimic the properties of layer IV of a rat somatosensory cortex. In the circuit, neurons are randomly positioned in three dimensions with an exponential connection probability that depends on the distance between the

neurons (see Table 2). To read the position of the ball, the excitatory neurons have *place fields* on the surface of the arena. The locations and distribution of the place fields are determined by logically stretching the two horizontal dimensions of the circuit to the size of the arena (see Figure 2). If the ball is near the place field of a neuron i, the neuron receives a current of the form:

$$I_i(\boldsymbol{r}) = I_{\max} \cdot \exp\left(-\frac{(\boldsymbol{r_i} - \boldsymbol{r})^2}{2\sigma^2}\right) . \tag{1}$$

Here, $I_{\max}$ denotes the maximum current, σ the size of a place field and $\boldsymbol{r_i}$ the position of the centre of the ith place field [8]. The response of the controller is generated by four *action cells* which integrate the activity of the excitatory neurons in the circuit. From the membrane potential $V_{\mathrm{m},j}$ of each action cell j we compute the force F_j according to:

$$F_j(V_{\mathrm{m},j}) = \frac{F_{\max}}{1 + \exp\left(-2 \cdot (V_{\mathrm{m},j} + 70\,\mathrm{mV})\right)} . \tag{2}$$

The details of the model are summarized in Table 2. The dynamics of ball and arena were simulated in Python. The neural controller was simulated with the Neural Simulation Tool NEST [6] .

3 Learning Paradigm and Experimental Results

To train the controller, we use a supervised learning algorithm that penalizes the difference between the optimal force $\boldsymbol{F_{\mathrm{opt}}}$ and the actually applied force. To avoid oscillations, the control must take the reaction time into account. Thus, given the position of the ball at time t, the network has to compute the required force at a later time $t + t_{\mathrm{f}}$.

The force is split into the four components $F_{\mathrm{opt},0}$ to $F_{\mathrm{opt},3}$ in the following way: Only those two force components are non-zero which are adjacent to the quadrant in which the ball will reside at time $t + t_{\mathrm{f}}$:

$$\begin{cases} (F_{\mathrm{opt},1} \text{ or } F_{\mathrm{opt},3}) = F_{\mathrm{abs}} \cdot \cos\phi \\ (F_{\mathrm{opt},0} \text{ or } F_{\mathrm{opt},2}) = F_{\mathrm{abs}} \cdot \sin\phi \end{cases} \tag{3}$$

with the absolute value of the force

$$F_{\mathrm{abs}} = F_{\max} \cdot \left(\frac{1}{1 + \exp\left(-16 \cdot (|\boldsymbol{r}_{\mathbf{guess}}| - 0.16\,\mathrm{m})\right)} - \frac{1}{15}\right) . \tag{4}$$

The vector $\boldsymbol{r}_{\mathbf{guess}}$ and the value ϕ are specified as

$$\boldsymbol{r}_{\mathbf{guess}}(t_{\mathrm{f}}) = \boldsymbol{r} + \boldsymbol{v} \cdot t_{\mathrm{f}} + \frac{1}{2} \cdot \boldsymbol{a} \cdot t_{\mathrm{f}}^2 , \tag{5}$$

and

$$\phi = \arctan\left|\frac{y_{\mathrm{guess}}}{x_{\mathrm{guess}}}\right| . \tag{6}$$

Here, $\boldsymbol{a}$ is the acceleration of the ball and $F_{\max}$ the force maximum. x_{guess} and y_{guess} are the components of the vector $\boldsymbol{r}_{\mathbf{guess}}$. The force in Equation 4 will be larger than the weight force of the ball, if the ball is outside its target field, and it will match the weight force, if the ball is inside the target field.

We applied the learning algorithm 1, to adjust the weights between the recurrent network (the liquid) and the action cells.

Algorithm 1. Learning algorithm

Require: $t = 1\,\text{ms}$
  while $t \leq t_{\text{learn}}$ **do**
    if t is integral multiple of T **then**
      for $i \in$ action cells **do**
        Compute $\bar{V}_{\text{m},i}\left([t-T,t]\right)$, $F_{\text{act},i}\left(\bar{V}_{\text{m},i}\right)$
        Compute $F_{\text{opt},i}$, $\Delta_i = F_{\text{opt},i} - F_{\text{act},i}$
        if $|\Delta_i| > \Delta_{\text{th}}$ **then**
          for $j \in$ place cells **do**
            if j spiked in $[t-T,t]$ **then**
              $\Delta w_{j\to i} = \gamma \cdot \Delta_i \cdot |l\,(i)\,|$
    $t = t + 1\,\text{ms}$

Here, T is the learning period and Δ_{th} is the maximum tolerable error. $\bar{V}_{\text{m},i}\left([t-T,t]\right)$ is the mean potential of action cell i over the latest period T. $w_{j\to i}$ denotes the conductance of the connection from neuron j to action cell i. The coefficient γ is the learning rate, which we kept constant. The factor $l(i)$ corresponds to the y-coordinate of the position of the ball if i is even, else to the x-coordinate. Near the target field, $l(i)$ provides a small weight adjustment. Otherwise, it reduces the time when the ball stays near the border of the arena. Table 1 summarizes the values of the task variables and the learning parameters.

Table 1. Task and learning parameters

Name	Value	Name	Value	Name	Value	Value
$F_{\max}$ [N]	7.2	$\alpha_{\max}/\beta_{\max}$ [°]	5.0	t_{f} [ms]	200.0	$I_{\max}$ [pA]	500.0
m_{ball} [g]	264.0	r_{arena} [cm]	50.0	T [ms]	20.0	σ [cm]	2.5
r_{force} [cm]	25.0	η	0.08	γ [$\frac{\text{nS}}{\text{Nm}}$]	1.0	t_{learn} [s]	10.0
r_{ball} [cm]	2.0	ρ	0.6	Δ_{th} [N]	0.1	t_{test} [s]	20.0

Each experiment is divided into training runs of duration t_{learn} and test runs of duration t_{test}. At the beginning of a run (training and testing), the ball is placed at a random position $\boldsymbol{r}_0$ with a random velocity $\boldsymbol{v}_0$ lower than $v = 0.3\,\frac{\text{m}}{\text{s}}$. After that, the trajectory of the ball is determined by the dynamics of the arena and the forces $F_{\text{act},0}$ to $F_{\text{act},3}$ applied by the network. There are five learning runs for each test run.

Figure 3 shows the results after 2500 seconds of learning, when no further improvement of the performance could be observed. Panel A shows the root mean square error for each component of the applied force. Each point denotes the result of a test run after the respective time of training. From $t = 350\,\mathrm{s}$ on, the the error signal reaches the value $\bar{\Delta}_i = (0.27 \pm 0.11)$ N, which is sufficient to steer the ball into the target field.

High errors between $t = 1000\,\mathrm{s}$ and $t = 2000\,\mathrm{s}$ in Figure 3A occur if the test run starts at a position which has not been trained before. Figure 3B shows the final position of the ball in a test run as a function of the learning time. Except for the failed runs, the ball stays close to the target field ($\bar{r}_{\mathrm{end}} = (6.7 \pm 2.8)$ cm).

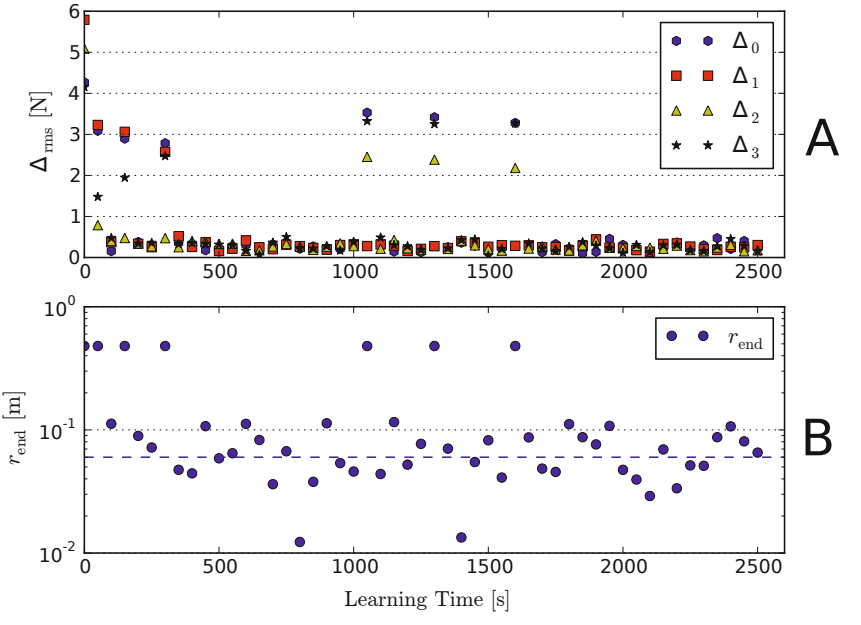

Fig. 3. Results of a learning session. Each measurement denotes the result of a test run after the indicated learning time. **(A)** The root mean squared error for each force component Δ_i as a function of the learning time. **(B)** Final position of the ball in a test trial as a function of the learning time. The *blue dashed line* shows the border of the target field.

Table 2. Model description according to [7]. Errors are given as standard deviations.

A: Model Summary	
Populations	2 populations: circuit, action cells; **circuit**: 1496 excitatory neurons, 501 inhibitory neurons; **action cells**: 4 excitatory neurons
Topology	**circuit**: superposed Gaussian minicolumns; mean distance between neurons: $\bar{d} = (285.0 \pm 123.6)\ \mu$m
Connectivity	**circuit**: distance-dependent connection probability following $p(r) = \frac{1}{8\pi \cdot (73.0\,\mu\text{m})^3} \cdot \exp\left(-\frac{r}{73.0\,\mu\text{m}}\right)$ which results from the geometrical arrangement of the neuronal morphologies, see E for connectivity matrix; **action cells**: no interconnections; **place cells to action cells**: all-to-one-connections, connection strengths follow a Gaussian distribution $\mathcal{N}(\mu = 0.0\,\text{nS},\ \sigma = 2.0\,\text{nS})$
Neuron model	**circuit**: adaptive exponential integrate-and-fire neurons, see B **action cells**: leaky integrate-and-fire neurons, see C
Synapse model	**all**: static conductance-based synapses, see D for synaptic statistics
Plasticity	supervised learning between excitatory neurons and action cells
Input	**circuit**: independent Poisson noise, rates: $\nu_\text{e} = 12000\,\text{Hz}$, $\nu_\text{i} = 200\,\text{Hz}$

B: Circuit Neuron Model (AdEx) according to [5]			
Name	**Value (exc.)**	**Value (inh.)**	**Description**
a [nS]	-5.17	-0.99	subthreshold adaptation
b [pA]	111.77	8.86	spike-triggered adaptation
V_T [mV]	-52.00	-57.00	spike threshold
Δ_T [mV]	2.00	2.00	slope factor
I_e [pA]	0.00	0.00	synaptic current
C [pF]	73.05	73.05	membrane capacitance
g_L [nS]	8.59	7.35	leak conductance
V_r [mV]	-74.35	-70.92	reset voltage
τ_w [ms]	55.27	1000.00	adaptation time constant
t_ref [ms]	5.00	5.00	absolute refractory time
V_peak [mV]	-30.00	-30.00	peak voltage
E_L [mV]	-74.35	-70.92	leak reversal potential
E_e [mV]	0.00	0.00	excitatory reversal potential
E_i [mV]	-70.00	-70.00	inhibitory reversal potential

C: Action Cell Model (LIF)		
Name	**Value**	**Description**
V_T [mV]	∞	spike threshold
I_e [pA]	0.0	synaptic current
C [pF]	250.0	membrane capacitance
V_r [mV]	-70.0	reset voltage
E_L [mV]	-70.0	leak reversal potential
τ_m [ms]	20.0	membrane time constant

D: Synaptic Statistics (circuit)		
Name	**Value**	**Description**
$w_e \pm \delta w_e$	(0.60 ± 0.22) nS	excitatory synaptic strength
$w_i \pm \delta w_i$	(1.92 ± 0.90) nS	inhibitory synaptic strength
$d_e \pm \delta d_e$	(1.09 ± 0.66) ms	excitatory synaptic transmission delays
$d_i \pm \delta d_i$	(1.08 ± 0.45) ms	inhibitory synaptic transmission delays

E: Connection Probabilities (circuit)			
		to	
		Excitatory	Inhibitory
from	Excitatory	0.195	0.129
	Inhibitory	0.272	0.071

4 Discussion

In this paper, we presented a neural controller based on spiking neurons and liquid computing that solves a dynamic control task. Using a supervised learning rule that penalizes the error between the applied force and the required force, we adapt the weights to the four action cells. The input to the controller encodes the position of the ball at time t. The output is the force vector that controls the table. After learning, the system manages to keep the ball within or very close to the target field. Since the force output is produced with some delay t_{f}, the performance is limited by the ability of the network to predict the position and velocity of the ball at the time when the force is applied.

Another important aspect of the system is that the time scale of the control task and the time scales of the controller must be compatible. The time scale of the controller is determined by the time constants of neurons and the connection delays, which are determined by biological processes and are beyond our control. The time scale of the control task mainly depends on the maximal inclination angle of the arena and the friction between the arena and the ball.

Acknowledgements. Partial funding was provided from the European Union project FP7-269921 (BrainScaleS).

References

1. Joshi, P., Maass, W.: Movement Generation with Circuits of Spiking Neurons. Neural Computation 17, 1715–1738 (2005)
2. Burgsteiner, H.: Training Networks of Biological Realistic Spiking Neurons for Real-Time Robot Control (2005)
3. Bouganis, A., Shanahan, M.: Training a Spiking Neural Network to Control a 4-DoF Robotic Arm based on Spike Timing-Dependent Plasticity. In: IEEE World Congress on Computational Intelligence, Barcelona, pp. 4104–4111 (2010)
4. Maass, W., Natschlaeger, T., Markram, H.: Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. Neural Computation 14(11), 2531–2560 (2002)
5. Brette, R., Gerstner, W.: Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity. J. Neurophysiol. 94, 3637–3642 (2005)
6. Gewaltig, M.-O., Diesmann, M.: NEST (Neural Simulation Tool). Scholarpedia 2(4), 1430 (2007)
7. Nordlie, E., Gewaltig, M.-O., Plesser, H.E.: Towards Reproducible Descriptions of Neuronal Network Models. PLoS Comput. Biol. 5(8), e1000456 (2009)
8. Vasilaki, E., Frémaux, N., Urbanczik, R., Senn, W., Gerstner, W.: Spike-Based Reinforcement Learning in Continuous State and Action Space: When Policy Gradient Methods Fail. PLoS Compuatational Biology 5(12), e1000586 (2009)