Evolution of Spiking Neural Circuits in Autonomous Mobile Robots

Dario Floreano, Yann Epars, Jean-Christophe Zufferey, Claudio Mattiussi

The authors are with the Autonomous Systems Laboratory, Institute of Systems Engineering, Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland. E-mail: Name.Surname@epfl.ch

Abstract

We describe evolution of spiking neural architectures to control navigation of autonomous mobile robots. Experimental results with simple fitness functions indicate that evolution can rapidly generate spiking circuits capable of navigating in textured environments with simple genetic representations that encode only the presence or absence of synaptic connections. Building on those results, we then describe a low-level implementation of evolutionary spiking circuits in tiny micro-controllers that capitalizes on compact genetic encoding and digital aspects of spiking neurons. The implementation is validated on a sugar-cube robot capable of developing functional spiking circuits for collision-free navigation.

I. Spiking Neural Circuits

The great majority of biological neurons communicate using self-propagating electrical pulses called *spikes*. Computational approaches to the study of brain function define two classes of neuron models that, among other things, differ in their interpretation of the role of spikes. Connectionist models [23], by far the most widespread, assume that what matters in the communication is the *firing rate* of a neuron, that is, the average quantity of spikes emitted by the neuron within a relatively long time window (for example, over 100 ms). In these models the real-value output of a neuron represents the firing rate, possibly normalized relatively to the maximum attainable value. Pulsed models [19], instead, are based on assumption that the *firing time*, that is, the precise time of emission of a single spike, may convey important information [25]. Often, these pulsed network models use complex activation functions that represent the emission of spikes on a very fine timescale [22].

Leaving aside the question of whether information transmitted among neurons is encoded by firing rate, firing time, or a combination of both, artificial spiking neural networks are attracting increased attention because they could capture and exploit more efficiently (i.e., with fewer neurons or with higher probability) non-linear time series of input signals; can be implemented in tiny and low-power chips [13] that exploit the sub-threshold physics of transistors in analog VLSI [20]; and allow biologically plausible investigations of computation in nervous systems. In this paper we are concerned mainly with the latter issue and show that adaptive networks of spiking neurons can be efficiently implemented also in tiny, low-cost, and largely available digital circuits.

Designing circuits of spiking neurons that display a desired functionality is still a challenging task. The most successful results in the field of robotics obtained so far focused on the first stages of sensory processing and on relatively simple motor control. For example, Indiveri et al. [12] developed neuromorphic vision circuits that emulate interconnections among neurons in the early layers of the biological retina in order to extract motion information and implement a simple form of attentive selection. These vision circuits have been interfaced with a Koala robot and their output has been used to drive the wheels of the robot in order to follow lines [14]. In another line of work, Lewis et al. developed an analog VLSI circuit with four spiking neurons capable of controlling a robotic leg and adapting the motor commands using sensory feedback [18]. This neuromorphic circuit consumes less than 1 microwatt and takes less than 0.4 square millimeters of chip area.

Despite these promising implementations, there are not yet methods for developing complex spiking circuits that could display minimally-cognitive functions or learn behavioral abilities through autonomous interaction with a physical environment. Artificial evolution thus may represent a promising methodology to generate networks of spiking circuits with desired functionalities expressed as behavioral criteria (fitness function). In previous work [4], we showed that evolution of spiking circuits can generate functional networks of spiking circuits for vision-based navigation of autonomous robots. Neuro-ethological analysis of an evolved circuit revealed functional specialization of single neurons and the role of spiking correlation on behavior. More recently, DiPaolo [3] used a similar approach to investigate the role of noise and synaptic plasticity in light-directed tasks.

In this article, we expand our previous work [7], [4] and describe a compact digital implementation of evolutionary spiking circuits that capitalize on our findings that such circuits do not require specification of synaptic weights and thus result in compact genetic encodings. The resulting evolutionary spiking circuit on chip, which occupies less than 50 bytes of memory, is validated on a sugar-cube robot that autonomously and reliably develops the ability to navigate around a maze in a less than an hour. A preliminary implementation of this model was described in [7]. Here we describe a final implementation, a new set of experiments, and the analysis of evolved network architectures.

In the next section we describe the network architecture and genetic representation used in these experiments. In the section that follows we briefly describe a set of evolutionary experiments on vision-based navigation with a neuron model that captures non-linear dynamics of synaptic integration and post-spike membrane behavior. These experiments are based on the specifications that we presented in [4]. We then describe the implementation in a micro-controller of a simplified neuron model and evolutionary algorithm and present a set of evolutionary experiments with a fully autonomous sugar-cube robot. Finally, we discuss the relationship between our low-level digital implementation and other analog VLSI implementations of spiking networks, as well as scalability issues and extensions of our model.

II. Network Architecture and Genetic Representation

In this section we describe the architecture and genetic representation of evolutionary spiking neurons, which is common to all experiments presented in this article.

The number of neurons and sensors is predefined and cannot be changed by the evolutionary process. Only the connectivity pattern and neuron signs are genetically encoded and evolved. A network is composed of n neurons and s sensory neurons (figure 1).

Each neuron can receive connections from all neurons (including itself) and from all sensory receptors. A neuron can be excitatory or inhibitory and all outgoing connections have the same sign. Synaptic connections have weight $w = 1$ and their signs are determined by the pre-synaptic neuron (positive if the neuron is excitatory, negative if the neuron is inhibitory). The state of a neuron is described by its *membrane potential.* Incoming spikes affect the membrane potential; we will assume that excitatory spikes increase its value and inhibitory spikes decrease it. In the absence of input activity the membrane potential tends towards a resting value (this process is also known as *leakage*). When the membrane potential exceeds its firing threshold, the neuron emits a spike. Following a spike, the membrane potential is lowered to a negative value from which it gradually returns to its resting potential. This hinders the emission of a new spike within the time interval that immediately follows the firing event. This time interval is also known as *refractory period.*

Sensory neurons are not connected among themselves and are always excitatory. At each time interval, they emit a spike with a probability proportional to the response of

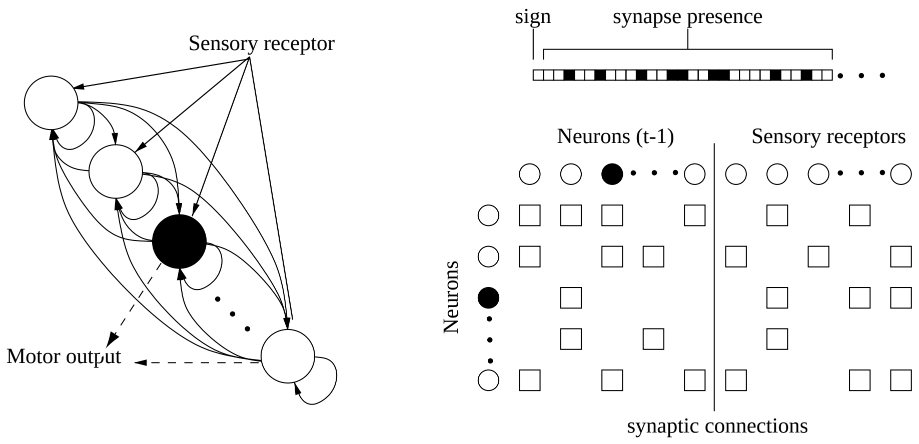


Fig. 1. Network architecture (only a few neurons and connections are shown) and genetic representation for one neuron. *Left*: A conventional representation showing the network architecture. White circles represent excitatory neurons, black circles represent inhibitory ones. *Right*: The same network unfolded in time. The circles on the top row represent the neurons and sensory receptors at a given time step; the circles on the left column represent the neurons at the next time step. The array of squares represent existing connections between neurons and from receptors to neurons. *Top right*: Genetic representation of one neuron. The neuron sign (excitatory or inhibitory) and the connectivity array are genetically encoded as 1's (excitatory neuron, connection, respectively) and 0's (inhibitory neuron, no connection, respectively). For every neuron, the first bit represents the neuron sign and the remaining bits represent the presence/absence of its incoming connections.

the corresponding sensor. The response of a sensor is linearly scaled in the interval $[0, 1]$.

A binary genetic string encodes the sign of each neuron and the presence of synaptic connections. All other neuronal and synaptic parameters are predefined and equal for all neurons. The genetic string is composed of n blocks, one for each neuron in the network. The first bit of the block encodes the sign of the neuron and the remaining $n + s$ bits encode the presence/absence of a connection from the n neurons and from the s sensory neurons in the network. Therefore, the total genetic length is $n(1 + n + s)$ bits.

III. Evolution of Vision-Based Navigation

In this first set of experiments we assess the evolvability of connectivity patterns (presence/absence of a connection) and neuron signs of fully recurrent spiking networks for a vision-based navigation task (preliminary results and additional experimental conditions are described in [4]).

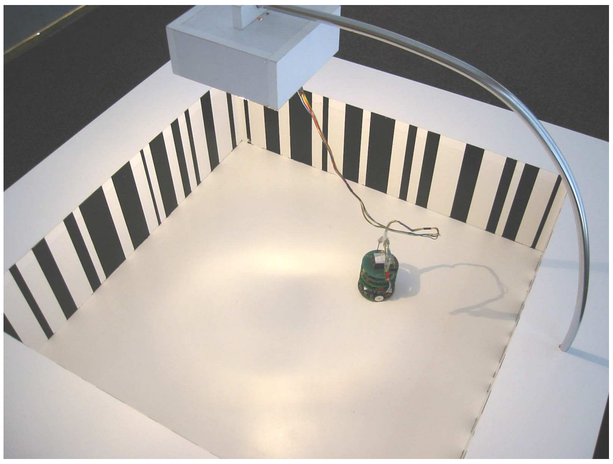


Fig. 2. A Khepera robot equipped with a linear camera is positioned in an arena with black and white vertical stripes of random size painted on the walls at irregular intervals. The arena is lit from above in order to let the evolutionary experiments continue at night. The robot is connected to a workstation through rotating contacts that provide serial data transmission and power supply. The spiking networks and genetic operators run on the workstation. The robot communicates with the workstation every 100 ms.

A Khepera robot equipped with a linear camera is asked to navigate in a square arena measuring 60 by 60 cm with textured walls (figure 2). The walls are filled with black and white vertical stripes. Width and spacing of the stripes have a uniform random distribution within the interval $[0.5, 5]$ cm.

The vision system (figure 3) is composed of a linear array of 64 photoreceptors (left hole) spanning a visual field of 36 deg and of a light sensor (right hole) used to adjust the sensitivity of the receptors to the global illumination level. Each photoreceptor returns a value between 0 (black) and 255 (white). Given the relatively low spatial frequency of the stripes on the walls, we read the activations of only 16 photoreceptors equally spaced on the array (1 every 4). These values are convolved with a Laplace filter spanning three adjacent (sampled) photoreceptors (weights of the Laplace filter are $\{-.5, 1, -.5\}$) in order to detect contrast (figure 3). Finally, the convolved image is rectified by taking the absolute values and scaling them in the range $[0, 1]$. The resulting 16 values represent the probabilities of emitting a spike for each corresponding sensory neuron at every update of the network. The states of all neurons in the network (including sensory neurons) are synchronously updated every millisecond, but the sensors and motors of the robot are updated only once

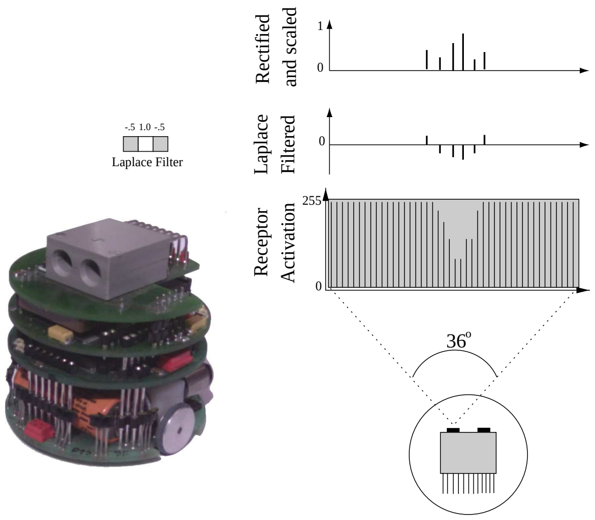


Fig. 3. The Khepera robot is equipped with a linear vision system composed of 64 photoreceptors. Only 16 photoreceptors are read every 100 ms and filtered through a Laplace filter in order to detect areas of contrast. The Laplace operator transforms the vector of values of receptor activation into a vector representing the values of the ”sources” of the variation in the activation of adjacent receptors, thus extracting the contrast information from the vector of receptor activations. The filtered values are transformed into positive values and scaled in the range $[0, 1]$. These values represent the probability of emitting a spike for each corresponding sensory neuron.

every 100 ms. During this interval, the spiking probability of sensory neurons corresponds to the most recent value returned by the robot. In addition to the 16 visual neurons, there is a bias neuron that is always active and can be used by evolution to determine a basic level of activity in the network in the absence of input-generated activity.

The network consists of 10 neurons that can be connected to each other and to all sensory neurons (figure 1). The number of spikes emitted by four motor neurons within the last 20 ms of the sensory-motor interval (100 ms) is used to set the speeds of the two wheels in push-pull mode. Each wheel of the robot is coupled to two neurons. The firing rate (number of spikes fired within 20 ms divided by maximum number of spikes) of one neuron is mapped into forward speed and the firing rate of the other neuron is mapped into backward speed. The sum of these two direction-specific speeds gives the final direction of rotation and speed of the wheel. Each wheel can take a maximum rotational speed of 80 mm/s, which would be obtained for a firing rate corresponding to the production of a spike at each update cycle of the network. However, since a neuron can fire at maximum once every two update cycles (because of the refractory period), the maximum speed is 40 mm/s.

In this set of experiments, we chose the *Spike Response Model* [8] of spiking neurons because the model is relatively simple and encompasses a large class of spiking neurons, including the simplified model that will be described later for the micro-controller implementation. In what follows, we describe the model and give between brackets the parameter values used in this experiment. In the Spike Response Model, the membrane potential $v_i(t)$ of a neuron i at time t is obtained by adding two kernels - one, $\epsilon(s)$, describing the effect of incoming spikes, and one, $\eta(s)$, describing the refractory period - as follows

$$v_i(t) = \sum_j w_j \sum_f \epsilon_j(s_j) + \sum_f \eta_i(s_i) \tag{1}$$

where $s_n = t - t_n^f$ is the difference between current time t and the firing time t^f of neuron n, and w_j (1 for all synapses) is the synaptic strength of the connection from neuron j. If the membrane potential $v_i(t)$ exceeds the neuron threshold θ_i (0.1 for all neurons), the neuron emits a spike and the corresponding time instant is added to the set of firing times. In these experiments

The properties of the kernel $\epsilon(s)$ are specified by *a)* the delay Δ (2 ms for all synapses) between the generation of a spike at the pre-synaptic neuron and the time of arrival at the synapse, *b)* a synaptic time constant τ_s (10 ms for all synapses), and *c)* a membrane time constant τ_m (4 ms for all synapses). A possible function $\epsilon(s)$ describing this behavior [9] is given by

$$\epsilon(s) = \exp[-(s-\Delta)/\tau_m](1 - \exp[-(s-\Delta)/\tau_s]) \tag{2}$$

for $\Delta \leq s \leq 20$, otherwise $\epsilon(s) = 0$.

The refractory period depends only on the membrane time constant τ_m. A possible kernel $\eta(s)$ [9] is given by

$$\eta(s) = -\exp[-s/\tau_m] \tag{3}$$

The value returned by $\eta(s)$ is weighted by a random value with uniform distribution in the range $[0, 1]$ in order to break ties in a network of interconnected neurons and prevent spontaneous emergence of locked oscillations.

In this set of experiments we use a generational, fixed population size, genetic algorithm [10]. A population of 60 individuals is evolved using rank-based truncated selection (15 best individuals, each generating 4 offspring), one-point crossover ($p = 0.1$ per pair), bit mutation ($p = 0.05$ per bit), and elitism (size=1).

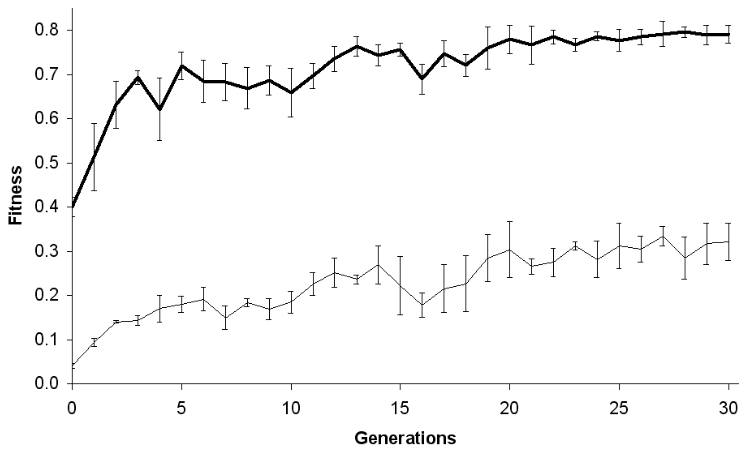
Each individual of the population is decoded and tested on the robot *two times* for 40 seconds each (400 sensory-motor steps). The robot is not repositioned between trials of the same individual or between different individuals. The fitness function Φ is the sum of the speeds of the two wheels v_{left} and v_{right} measured by the optical encoders at every time step t (100 ms), only if both wheels rotate in the forward direction, averaged over T time steps available (here $T = 400 + 400$)

$$\Phi = \frac{1}{T} \sum_{t}^{T} (v_{left}^{t} + v_{right}^{t}) \tag{4}$$

If v_{left} or v_{right} are less than 0 (backward rotation) or equal to 0 (no rotation), $\Phi^t = 0$. This fitness function selects individuals for the ability to go as straight and as fast as possible. In addition, since the robot takes only a few seconds to travel across the arena and wheels rotate considerably less if the robot is stuck against a wall, the fitness function implicitly encourages selective reproduction of individuals that can avoid walls. The fitness function does not use the active infrared sensors available on the robot to measure distance from the walls (as we did in previous experimental work [6, e.g.]) because the response profile of these sensors varies depending on the reflection properties of the walls (black stripes reflect approximately 40% less infrared light than white stripes) and on the infrared spectrum component of ambient illumination.

In this set of experiments, the neural network, evolutionary algorithm, and fitness computation are implemented on a desktop PC connected to the robot through the serial port and rotating contacts, which provide also energy supply. For a description of the methodology, see [21, chapter 3].

We have run three experiments on the physical robot. Each experiment starts with a different random initialization of the genetic strings. One generation on the physical robot took 80 minutes. In all runs, artificial evolution took less than 30 generations to discover spiking controllers capable of navigating around the environment and avoiding the walls. The graph on the left of figure 4 displays population mean and population best



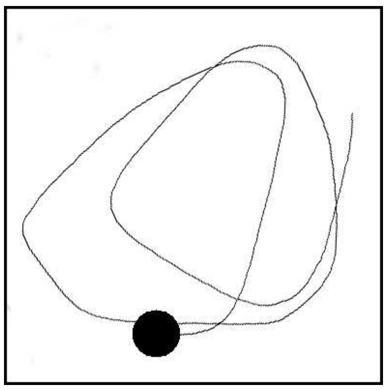


Fig. 4. *Left*: Fitness values obtained on the physical robot Khepera (best fitness = thick line; average fitness = thin line). Each data point is the average of three evolutionary runs with different random initializations. Bars indicate standard error. *Right*: Trajectory of en evolved individual. The plot is obtained by tracking the wheel rotations for an entire trial (40 s) and fitting the trajectory within the square arena. The black disk shows the position of the robot at the end of the trial.

fitness values averaged across three runs. Fitness values above 0.6 already correspond to robots that can move forward and avoid walls. Further fitness gains correspond to faster and smoother trajectories (an example is shown on the right side of figure 4). Fitness values of 1.0 cannot be reached because the robot sometimes must reduce the speed of one wheel in order to turn and avoid walls.

Since initial populations are randomly created, only 50% of the connections are present. This percentage did not change significantly along generations in any of the evolutionary runs. In [4] we described several methods of analysis and used them to understand an evolved spiking controller (evolved in a different arena from that used for the experiments described here). For the purpose of this paper, the most important result is that a compact genetic representation that describes only the pattern of connectivity and neuron sign is sufficient to evolve functional networks of spiking controllers. In the rest of this paper, we capitalize on this result to implement a simplified evolutionary spiking network in low-power microcontrollers.

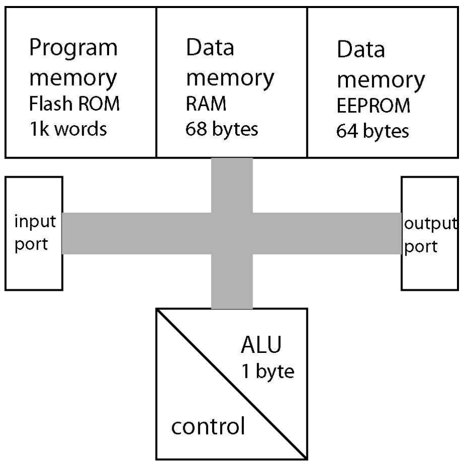


Fig. 5. Components of a microcontroller with von Neumann architecture (Values are given for the Microchip PIC16F268 microcontroller). The microprocessor unit is composed of an Arithmetic Logic Unit and of control devices to move data from/to memory banks and input/output ports. The memory banks are organized in physically separated locations. For example, the microcontroller shown in the figure uses the ROM memory to store a program composed of a maximum of 2k instructions; a RAM memory to store 224 bytes of data; and an EEPROM memory to store 128 bytes of data. The input/output ports can be connected to sensors, keyboards, LEDs, motorized actuators, or any other peripheral. Gray lines represent the bus where one instruction or data item at a time is moved across components.

IV. Evolutionary Spiking Circuits in a Microcontroller

A microcontroller is an integrated circuit composed of a microprocessor unit, memory, and input/output peripheral devices (figure 5). In other words, it is a full computer in a single chip capable of receiving, storing, processing, and transmitting signals to the external world. Most applications using microcontrollers require very low power consumption, small size, robustness to hard operating conditions, and low price. These features come at the expense of the number of transistors and instructions per second, resulting in very low computing power compared to personal computers. Consequently, low-level languages, such as assembler, are often used to exploit efficiently every single bit of memory.

The core idea explored in this paper is that spiking circuits can be mapped quite easily into microcontrollers because spikes are essentially binary events and the non-linear dynamics and neural information is given by spiking time and spike count, rather than by non-linear, real-valued, activation functions used in connectionist neuron models. In this implementation we use a few logic operations (such as AND, NOT, and bit shift) to implement a network of spiking neurons.

The experiments described in the previous section showed that artificial evolution can easily discover functional spiking circuits by exploring only the space of neuron sign and connectivity. Both variables can be described by a single bit (1 = positive sign for neurons, connection enabled for synapses; 0 = negative sign, connection disabled for synapses) and therefore can be efficiently stored and easily manipulated in microcontrollers.

The next two subsections will describe the neuron and evolutionary model, respectively. Implementation details are described in Appendix A and B. The section that follows will describe an example of this implementation where a microrobot equipped with a microcontroller evolves without human intervention in less than two hours the ability to move around a maze.

The chip used in the experiments described here belongs to the PIC (Peripheral Interface Controller) family of microcontrollers by Arizona Microchip Technology (www.microchip.com). However, the same implementation method is applicable to any other type of microcontroller.

A. Neuron Model and Implementation

The neuron model used in the experiments with the Khepera robot described above is much too complex to be implemented in a microcontroller because it uses several non-linear functions, requires floating-point representation and relatively high computing speed. Therefore, the neuron model used here is a simple integrate-and-fire model with leakage and refractory period.

The behavior of a neuron (figure 6) is described by the following series of steps:

1. *Refractory period.* If the neuron has emitted a spike within the previous time interval Δt, the membrane potential is not updated. In these experiments, $\Delta t = 1$.
2. The *contribution of incoming spikes* e_i^t is given by the sum of spikes o_j^t at time t through existing connections w_{ij} weighted by the sign of emitting neurons s_j:

$$e_i^t = \sum_j^N o_j^t w_{ij} s_j \tag{5}$$

where $o_j^t \in \{0, 1\}$, $w_{ij} \in \{0, 2\}$, $s_j \in \{-1, 1\}$.

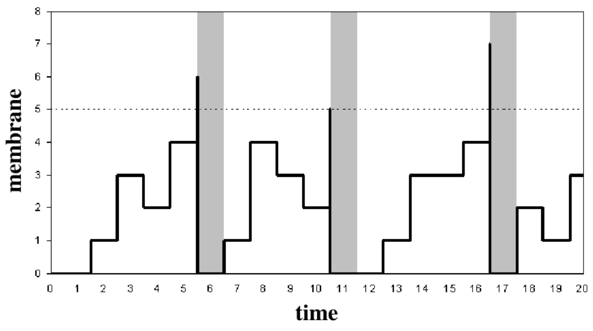


Fig. 6. Behavior of a neuron with constant firing threshold. Values are those used in the experiments described in this paper.

3. *The membrane potential* υ_i^t is updated by adding the contribution of incoming spikes to the available potential. If the result is lower than the resting potential υ_i^{min}, the membrane potential is set to the resting potential.

$$\upsilon_i^t = \begin{cases} \upsilon_i^{t-1} + e_i^t & \upsilon_i^{t-1} + e_i^t \geq \upsilon_i^{min} \\ \upsilon_i^{min} & \text{otherwise} \end{cases} \tag{6}$$

where $\upsilon_i^{min} = 0 \; \forall i$ in these experiments.

4. *Spike generation.* If the membrane potential is larger than, or equal to, a threshold υ_i^{max}, the output of the neuron is set to 1 (spike) and the membrane potential to its resting potential υ_i^{min}; otherwise the output of the neuron is set to 0 (no spike) and the membrane potential is not affected.

$$o_i^t = \begin{cases} 1 \text{ and } \upsilon_i^t = \upsilon_i^{min} & : \quad \upsilon_i^t > \upsilon_i^{max} + r^t \\ 0 & : \quad \text{otherwise} \end{cases} \tag{7}$$

where here the threshold $\upsilon_i^{max} = 5 \; \forall i$ and r^t is a random integer in the range $[-2, 2]$ to prevent the emergence of locked oscillations in networks with feedback connections.

5. *Leakage.* A leaking constant k_i is subtracted from the membrane potential only if the result of this operation is larger or equal to the resting potential υ_i^{min}

$$\upsilon_i^t = \begin{cases} \upsilon_i^t - k_i & : \quad \upsilon_i^t - k_i \geq \upsilon_i^{min} \\ \upsilon_i^{min} & : \quad \text{otherwise} \end{cases} \tag{8}$$

Here $k_i = 1 \; \forall i$.

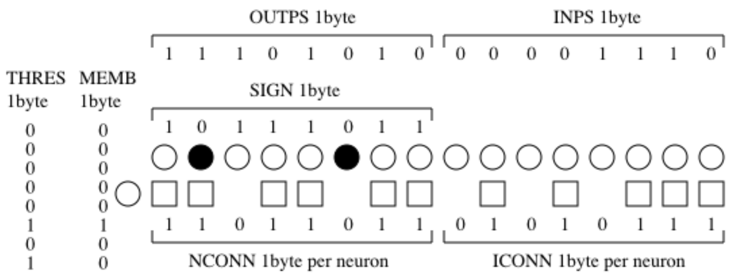


Fig. 7. Digital representation of one neuron in the microcontroller.

The circuit architecture is similar to that used for the experiments on vision-based navigation described above. Each neuron can be connected to all neurons (including itself) and to all sensory neurons, as in figure 1. The sign of the neuron determines the effect of its spikes on other neurons (equation 6). The presence of a spike in the sensory neuron is determined by the activity of sensors, as explained later.

The implementation (figure 7) exploits the 8-bit architecture of the microcontroller used in these experiments. Therefore, the network is composed of 8 neurons and 8 sensory neurons. At every network cycle, the spiking state of all neurons and sensory neurons are encoded by the byte OUTPS and INPS, respectively (a bit takes value 1 if the corresponding neuron emitted a spike at the previous cycle, otherwise is 0). The sign of all neurons is described by the byte SIGN (bit value is 1 if the corresponding neuron is excitatory, 0 if it is inhibitory). The pattern of incoming connections for one neuron is described by byte NCONN for connections from neurons and by byte ICONN for connections from sensory neurons. Each neuron has one byte MEMB to store its membrane potential. The threshold of all neurons is encoded by the byte THRES. This network requires 28 bytes of RAM memory (INPS, OUTPS, SIGN, THRES, 8 x MEMB, 8 x NCONN, 8 x ICONN). Nine additional bytes are used to store random numbers, counters, and temporary variables that are shared with the evolutionary algorithm described in the next subsection.

The update of a neuron is partly done in parallel by performing AND operations between the byte storing the spikes and the byte storing the connections from excitatory neurons. The resulting number of active bits are used to increment the membrane potential of the neuron. Contributions from inhibitory neurons are computed in a similar fashion after taking the complement (NOT) of the byte storing the sign of all neurons and combining it with the pattern of connections. The resulting number of active bits is used to decrement the membrane potential. Network architectures of less than 8 neurons and/or sensory neurons can easily be implemented by masking (with AND) unused bits with a byte with bit value 1 for every used neuron. Details of the implementation are given in Appendix A.

B. Evolution Model and Implementation

The same genetic encoding used for the experiments with the Khepera (see figure 1) has been used for the neuron model used here. Consequently, the genetic string of the spiking circuit consists of only 17 bytes: 1 byte for the sign of the neurons (SIGN), 8 bytes for its neural connections (NCONN), and 8 bytes for its sensory connections (ICONN). An additional byte is used to store the fitness of the individual.

The memory constraints of microcontrollers puts a severe limit on the number of genetic strings (individuals) maintained in the population. Therefore, a form of steady-state genetic algorithm, which experimentally showed to be suitable for small populations [26], [24], has been chosen. The algorithm used here, designed to maximize exploration while preserving the best solution obtained so far, works as follows:

1. Randomly generate a population of genetic strings and initialize their fitness values to zero.
2. Pick an individual at random from the population, mutate it, and measure its fitness.
3. If its fitness is equal or larger to the fitness of the worst individual in the population, write its genetic string and fitness value at the memory location of the worst individual, otherwise throw it away.
4. Go to step 2.

Mutated individuals are put back in the population even if they have the same fitness of the worst individual in order to allow for “neutral walks” [17] on the genetic landscape. This may be a useful property for evolution of small converged populations [11]. Implementation details of this evolutionary algorithm in the microcontroller are given in Appendix B.

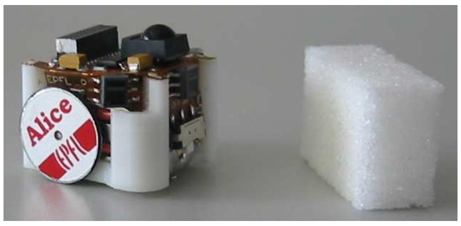


Fig. 8. The sugar-cube robot Alice. Four active infrared sensors are used to detect distance from obstacles within a 3 cm range. Three sensors are located in front of the robot (front, front left, and front right) and one on the back.

V. Embedded evolution of Micro-robot control

The method described above has been tested on a simple evolutionary task for an autonomous micro-robot equipped with a PIC microcontroller. Alice (figure 8) is one of the smallest autonomous mobile robots in the world [1] with an energetic autonomy of 10 hours. Alice is a programmable and modular robot. It measures approx. 2 cm on each side and has a weight of 10 g. In its basic configuration it has 2 bi-directional Swatch motors that allow a maximum speed of 40 mm/s, 4 active infrared sensors for detection of distance from obstacles, a PIC16F628 microcontroller at 4MHz, and a NiMH rechargeable battery. The infrared sensors have a limited range of 2 to 3 cm, which is similar to the size of the robot itself. Since the sensor output is noisy, the less significant bit of the A/D converter is used to re-initialize every 50 ms the pseudo-random number generator required to initialise the genetic strings, add noise to the neuron thresholds, and perform genetic mutations.

The robot is asked to navigate in a 25 by 18 cm white arena with a wall in the middle. The fitness is computed and accumulated at each sensory-motor cycle using a truncated version of a three-component function to evolve straight navigation and obstacle avoidance [5]

$$\Phi = (V)(1 - \Delta V)(1 - i)$$

where V is the sum of the speeds of the two wheels (this component is maximized by high wheel rotation), ΔV is the absolute difference between the two wheel-speeds (this component is maximized by straight navigation) and i is the activity of the most active sensor (this component is maximized by distance from obstacles). Since the Alice robot

sensor value	bits set
0-1	000
2-3	001
4	011
5-7	111

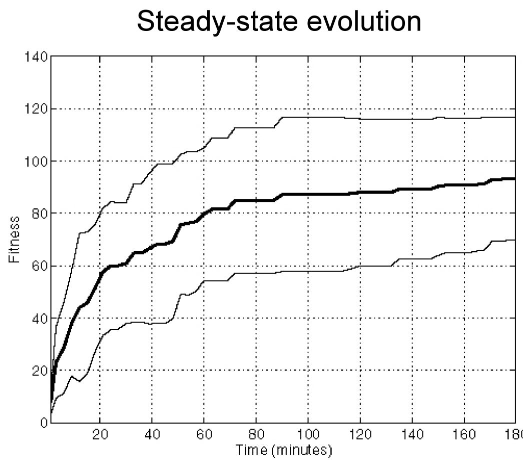
TABLE I

Coding of the sensory inputs.

does not have wheel encoders to measure wheel rotation, the speed values used in the fitness function are taken from the motor output of the neural circuit. Motor output is a good approximation of wheel speed except for the situation when the robot is against an obstacle (in that case the actual wheel velocity is zero or significantly lower than the motor output). However, in that condition the fitness returns a zero value because at least one of the infrared sensors has maximum activation. The function is truncated by setting its value to zero whenever one of the wheel speeds is in backward rotation. Each of the three terms is scaled so that the maximal fitness value of each sensory-motor cycle multiplied by the total number of cycles in a navigation trial could fit in a single byte.

The network architecture is composed of 8 neurons and 8 sensory units. Since the fitness function returns non-zero values only for forward navigation, the infrared sensor on the back of the robot is not used. The activations of the three frontal sensors are scaled in the range $[0, 7]$ and coded on three bits by setting active bits proportionally to the sensor activation, as shown in table V. Since the sensors tend to saturate when the robot is close to an obstacle, this bit encoding gives less precision for high sensor activation. Three sensory neurons are allocated for the front left and for the front right sensor each and two sensory neurons for the front sensor. The spiking state of each sensory neuron is given by the value of the corresponding bit using the lookup table V. Only the first two bits in the lookup table are used for the two neurons corresponding to the front sensor.

Sensors and motors are updated every 20 ms, but the spiking network is updated every 1.2 ms with the embedded R/C clock at 4 MHz. The rotation speed and direction of the



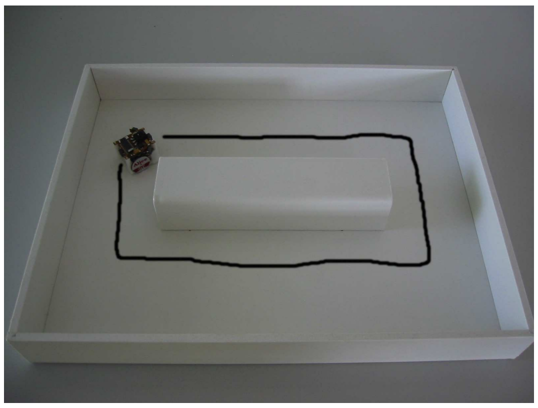


Fig. 9. *Left*: Minimum, average, and maximum fitness of best individuals in 6 evolutionary runs. Data points are sampled every 3 minutes. *Right*: Trajectory (over 11 s) of the best evolved individual. The path covered by the robot is taken from a video clip downloadable from `http://asl.epfl.ch`.

wheels is computed using the spike count of four motor neurons in push/pull mode, as for the experiments with the Khepera robot described above.

For each experiment, a population of 6 individuals was randomly initialized and evolved for 3 hours using on-board batteries. Each individual is tested for 14 seconds. Every three minutes the best fitness obtained so far was logged in a block of 60 bytes in the RAM and then downloaded to a computer at the end of the experiment. The graph on the left side of figure 9 shows the fitness values of the best individuals for five experiments with different random initialization of the population. A fitness value of 60 corresponds to a collision-free navigation for 14 seconds. Higher values are obtained by straight and faster trajectories. The best individual shown on the right side of figure 9 covers the entire arena in 11 s.

All best evolved individuals perform wall following around the obstacle in the middle of the arena while maintaining a distance that generates the lowest sensor activation. Figure 10 shows the architecture of the controller corresponding to the trajectory depicted on the right side of figure 9. This pattern of diagonal connectivity is found in almost all best evolved networks (with some individual variations). Neurons tend to have connections from a small set (3 on average) of neighboring sensors and from a small set (4 on average) of neighboring neurons. This pattern of connectivity loosely reminds topological sensory

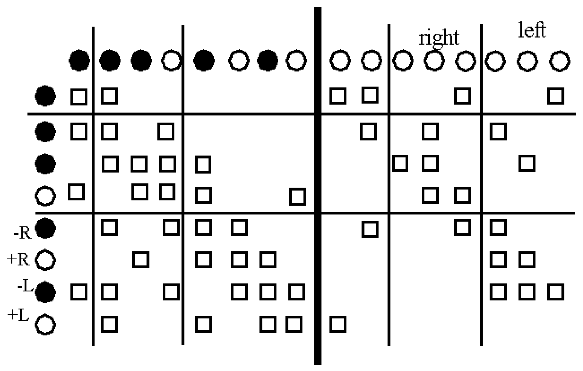


Fig. 10. Network architecture of an evolved individual. The same graphic conventions explained in figure 1 are used here. Sensory neurons are connected to front (2), right (3), and left (3) distance sensors on the robot, as explained in table V. Four neurons are used to set the speeds of the right and left wheels in push-pull mode, as explained in section III.

maps of biological brains where neighboring neurons receive activation from neighbouring sensors [15]. This layout ensures that smooth change in sensor space translates into smooth change in neural space and that small variations in sensory stimulation (for example, due to the movement of the agent) do not cause completely different patterns of neural activation.

VI. Discussion

In this article we have shown that artificial evolution is a suitable method to generate functional architectures of spiking neurons by searching only through the space of neuron sign and connectivity. The Spike Response Model used in the first set of experiments contains several parameters whose values have been taken from previous literature [9] and were not optimized for this specific implementation. Therefore, we cannot exclude that different parameter values would make the circuits harder or easier to evolve.

The only critical modification that we made to the Spike Response Model was the insertion of noise in the refractory period. In preliminary experiments without noise, evolution stagnated very quickly into poor systems because most of the neural circuits fell into locked oscillations that were not sensitive to sensory input and fitness values did not increase over generations. Noise in the refractory period anticipates or delays the firing time of neurons, thus decreasing the probability of locked oscillations generated by feedback loops. A similar effect can be obtained by adding some noise centered around zero to the membrane thresholds. This latter option was used in the micro-controller implementation of the simple spiking neuron because it required comparatively fewer resources.

The micro-controller implementation maintains the main features of the evolutionary spiking system, such as the genetic encoding and network architecture, but it introduces major simplifications in the evolutionary and neural algorithms. Although the experimental results described here are promising, we cannot exclude that the micro-controller system may have less computational abilities than the more complex model. The major difference between the Spike Response Model and the simpler digital model is that the former includes non-linear functions for synaptic signal transmission and refractory period. However, it is hard to tell what environmental and/or behavioral situations require those non-linearity.

We can finally compare the low-level spiking network implementation presented here, with analog VLSI implementations of comparable functionalities. It is clear that our microcontroller implementation must pay the price of programmability [2], that is, we can expect to achieve higher power consumption, lower speed and less computational parallelism than with an analog implementation that uses the same silicon resources. These drawbacks, however, have a counterbalance in the greater flexibility of a programmable implementation in the definition and adaptation of the circuit topology and parameters. In this respect, our low-level implementation, by exploiting the microcontroller parallelism and adopting an atomic representation of spikes as bits, goes in the direction of an optimal exploitation of the resources available in a programmable device. Furthermore adding learning rules into such a spiking network would require time-varying memory units. The most obvious implementation of such units in analog devices is by mean of capacitors, which are known to be space consuming and to suffer from leakage, whereas in microcontrollers it is straightforward to allocate more memory and processing resources to a run-time adaptive mechanism.

VII. Conclusion

We have described a simple method to evolve functional networks of spiking neurons, a low-level efficient implementation in microcontrollers, and experimental tests on two robot navigation problems.

These results could pave the way to two types of future developments. On the one hand, the method could be extended to study issues of information coding in networks of spiking neurons coupled to real environments. For example, one could investigate under what environmental, behavioral, and/or architectural conditions evolved spiking controllers rely on precise firing time rather than on firing rate. On the other hand, the micro-controller implementation could make its way in a number of embedded application that require adaptive signal processing. More than 3.5 billion microcontroller units are sold each year for embedded systems (washing machines, credit cards, car electronics, etc.), exceeding by more than an order of magnitude the number of microprocessor units sold for computers [16].

Appendix A: Micro-controller Implementation of the Spiking Network

The steps of the neuron model described above are implemented as follows:

1. *Refractory period.* Check state of corresponding bit in OUTPS; if set to 1, go to step 3.
2. Compute *contribution of incoming spikes* and *membrane update.* Start with spikes from sensory neurons: Increment MEMB variable by counting (left shift with carry) the number of active bits that result from the `AND` function of byte INPS and byte ICONN. Continue with spikes from positive neurons: Increment MEMB variable by counting the number of active bits that result from the `AND` function of bytes OUTPS, SIGN, and NCONN. Finish with spikes from negative neurons: Decrement MEMB variable by counting the number of active bits that result from the `AND` function of OUTPS and the complement (NOT function) of byte SIGN and byte NCONN. The decrement is stopped before MEMB goes below zero (which is signalled by a bit flag in a housekeeping byte of the microcontroller; this same byte also signals overflow, which does not occur here because there are few neurons in the network).
3. *Spike generation.* Compute random value for r_i and check whether MEMB is equal or larger to THRES incremented/decreased by r_i. If so (spike), set the corresponding bit in OUTPS to 1 and reset MEMB to zero. Otherwise (no spike), set corresponding bit in OUTPS to 0.
4. *Leakage.* If MEMB is greater or equal than the leaking constant $k_i = 1$, decrement it by the leaking constant $k_i = 1$.

The network is update synchronously, so that each neuron changes its state according to the state of all neurons computed at the previous cycle. Therefore, step 3 above updates only a temporary copy of OUTPS which is then moved into OUTPS once all neurons have been updated. Alternatively, one could update the network asynchronously by picking a neuron at random and changing directly OUTPS at step 3. Once the entire network has been updated, the array of sensory spikes INPS is updated too.

When run on a PIC16F628 using the embedded R/C oscillator running at 4MHz, the entire network is updated in approximately 1.2 ms. In some case, such as for the robotics experiment described here, the entire network can be updated faster than the time interval required to update sensors and motors (20 ms). Between new sensory values, INPS is set to all 0's while the neurons continue to be updated using only internally generated spikes.

Appendix B: Micro-controller implementation of the steady-state evolutionary algorithm

In these experiments, each individual is mutated at three locations by toggling the value of a randomly selected bit. The first mutation takes place in the SIGN byte that defines the signs of the neurons. The second mutation occurs at a random location of the NCONN block that defines the connectivity among neurons. The third mutation occurs at a random location of the ICONN block that defines the connectivity from sensors. Mutations are performed by making an XOR operation between the byte to be mutated and a byte with a single 1 at a random location.

The population (genetic strings and fitness values) is stored in the EEPROM because this type of memory can be read and written by the program just like the RAM memory, but in addition it holds its contents also when the microcontroller is not powered (at least 40 years for the microcontrollers used here). Each individual occupies a continuous block of bytes where the first byte is its fitness and the remaining 17 bytes represent the genetic string. The very first byte of the EEPROM memory records the number of replacements made so far. Whenever the microcontroller is powered up, the main program reads the first byte of the EEPROM. If it is 0, the population is initialized, otherwise it is incrementally evolved (step 2).

EEPROM memories can be written only a limited number of times (for example, the

EEPROM of the microcontroller used here can be written/read approximately 10,000,000 times) and usage and temperature generate errors during reading/writing (bit values are toggled) that require error-checking routines. Therefore, in the experiments described here, we keep a copy of the entire population in the free space of the RAM memory and copy it to the EEPROM only at predefined intervals.

Acknowledgements

This work was supported by the Swiss National Science Foundation, grant no. 620-58049. The authors thank Gilles Caprari for help with the interface between the evolutionary spiking circuit on chip and the Alice robot.

References

[1] G. Caprari, T. Estier, and R. Siegwart. Fascination of down scaling - alice the sugar cube robot. *Journal of Micro-Mechatronics*, 1(3):177–189, 2002.

[2] M. Conrad. The price of programmability. In R. Herken, editor, *The universal Turing machine. A half-century survey*, pages 285–307. Oxford University Press, Oxford, 1988.

[3] E. A. DiPaolo. Evolving spike-timing dependent plasticity for single-trial learning in robots. *Philosophical Transactions of the Royal Society A*, 361:2299–2319, 2003.

[4] D. Floreano and C. Mattiussi. Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots. In T. Gomi, editor, *Evolutionary Robotics. From Intelligent Robotics to Artificial Life*. Springer Verlag, Tokyo, 2001.

[5] D. Floreano and F. Mondada. Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot. In D. Cliff, P. Husbands, J. Meyer, and S. W. Wilson, editors, *From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 402–410. MIT Press-Bradford Books, Cambridge, MA, 1994.

[6] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:396–407, 1996.

[7] D. Floreano, C. Schoeni, G. Caprari, and J. Blynel. Evolutionary Bits'n'Spikes. In R. K. Standish, M. A. Bedau, and H. A. Abbass, editors, *Artificial Life VIII. Proceedings of the Eighth International Conference on Artificial Life*. MIT Press, Cambridge, MA, 2002.

[8] W. Gerstner. Associative memory in a network of biological neurons. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information processing Systems 3*, pages 84–90. Morgan Kaufmann, San Mateo, CA, 1991.

[9] W. Gerstner, J. L. van Hemmen, and J. D. Cowan. What matters in neuronal locking? *Neural Computation*, 8:1653–1676, 1996.

[10] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, Redwood City, CA, 1989.

[11] I. Harvey and A. Thompson. Through the labyrinth, evolution finds a way: A silicon ridge. In T. Higuchi, M. Iwata, and W. Liu, editors, *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware*. Springer Verlag, Tokyo, 1996.

[12] G. Indiveri. A Neuromorphic VLSI device for implementing 2D selective attention systems. *IEEE Transactions on Neural Networks*, page in press, 2001.

[13] G. Indiveri and R. Douglas. ROBOTIC VISION: Neuromorphic Vision Sensors. *Science*, 288:1189–1190, 2000.

[14] G. Indiveri and P. Verschure. Autonomous vehicle guidance using analog vlsi neuromorphic sensors. In W. Gerstner, A. Germond, M. Hasler, and J-D. Nicoud, editors, *Proceedings of the 7th International Conference on Neural Networks*, pages 811–816, Berlin, 1997. Springer Verlag.

[15] E. R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of Neural Science.* McGraw-Hill Professional Publishing, New York, 2000. 4th edition.

[16] S. Katzen. *The Quintessential PIC Microcontroller.* Springer Verlag, London, 2001.

[17] M. Kimura. *The Neutral Theory of Molecular Evolution.* Cambridge University Press, Cambridge, UK, 1983.

[18] M. A. Lewis, R. Etienne-Cummings, A. H. Cohen, and M. Hartmann. Toward biomorphic control using custom aVLSI CPG chips. In *Proceedings of IEEE International Conference on Robotics and Automation.* IEEE Press, 2000.

[19] W. Maas and C. M. Bishop, editors. *Pulsed Neural Networks.* MIT Press, Cambridge, MA, 1999.

[20] C. Mead. *Analog VLSI and Neural Systems.* Addison-Wesley, Reading, MA, 1989.

[21] S. Nolfi and D. Floreano. *Evolutionary Robotics: Biology, Intelligence, and Technology of Self-Organizing Machines.* MIT Press, Cambridge, MA, 2000.

[22] F. Rieke, D. Warland, R. van Steveninck, and W. Bialek. *Spikes. Exploring the neural code.* MIT Press, Cambridge, MA, 1997.

[23] D. E. Rumelhart, J. McClelland, and PDP Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations.* MIT Press-Bradford Books, Cambridge, MA, 1986.

[24] G. Syswerda. Uniform crossover in genetic algorithms. In *Proc. of the Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, San Mateo, CA, 1989.

[25] A. Villa. Empirical evidence about temporal structure in multi-unit recordings. In R. Miller, editor, *Time and the Brain.* Harwood Academic Publishers, Reading, UK, 2000.

[26] D. Whitley and J. Kauth. GENITOR: A different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, Denver, Colorado, 1988.