Evolutionary Bits'n'Spikes

Dario Floreano, Nicolas Schoeni, Gilles Caprari, Jesper Blynel*

Autonomous Systems Laboratory (asl.epfl.ch), Institute of Systems Engineering
Swiss Federal Institute of Technology (EPFL), CH-1015, Lausanne, Switzerland

Abstract

We describe a model and implementation of evolutionary spiking neurons for embedded microcontrollers with few bytes of memory and very low power consumption. The approach is tested with an autonomous microrobot of less than 1 in^3 that evolves the ability to move in a small maze without human intervention and external computers. Considering the very large diffusion, small size, and low cost of embedded microcontrollers, the approach described here could find its way in several intelligent devices with sensors and/or actuators, as well as in smart credit cards.

Artificial Spiking Circuits

Most biological neurons communicate by sending pulses across connections to other neurons. The pulse is also known as "spike" to indicate its short and transient nature. Neurons are affected by incoming spikes and generate a spike when their membrane potential (voltage or state) becomes larger than a threshold. Spike generation is followed by a short "refractory period" during which the neuron cannot generate another spike.

Computational models of spiking neurons are attracting increasing interest in engineering and computer science (Maas & Bishop 1999). On the one hand, computer simulations of spiking networks can help to address specific questions in neuroscience, such as how biological neurons communicate with each other (Koenig, Engel, & Singer 1996; Rieke *et al.* 1997). On the other hand, a better understanding of spiking neurons is leading to the development of new neuromorphic devices (Horiuchi 2001), some of which may replace lesioned fibers or sensory organs.

In addition, we argue that networks of spiking neurons represent suitable control systems for autonomous behavioral systems,[1] such as situated autonomous robots, because temporal patterns of sensory-motor events may be captured and exploited more efficiently (i.e., with fewer neurons or with higher probability) by the intrinsic time-dependent dynamics of spiking neurons than by other connectionist models (Rumelhart, McClelland, & PDP Group 1986).

Computational investigations of spiking neurons are often based on biophysical models that are significantly more complicated than models used in connectionism (McCulloch-Pitts, Perceptron, and Hopfield models, e.g.). These biophysical models are defined by coupled differential equations that attempt to capture the complex dynamics of membrane ion channels, generation and propagation of spikes, and synaptic conductances, among others. A representative sample of these models is presented in "road map" II.5 of the *Handbook of Brain Theory and Neural Networks* (Arbib 1998). Although such differential equations can be directly mapped into analog VLSI circuits exploiting the physics of sub-threshold transistors (Mead 1989; Indiveri & Douglas 2000), digital implementations of spiking networks often resort to complex simulations (Bower & Beeman 1994) and/or comparatively larger hardware resources (de Garis 1996).

Whatever choice of implementation, hand-design of spiking circuits that display a desired functionality is not a trivial task because of the highly non-linear dynamics. Furthermore, the learning algorithms developed so far for spiking circuits are often restricted to very simple and application-specific architectures (Maas & Bishop 1999). 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.* (2001) 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

*Dario Floreano conceived the model and experiments described here, implemented the evolutionary spiking network in assembler language, and wrote most of this paper. Nicholas Schoeni ported the software to the microrobot and run the evolutionary experiments. Gilles Caprari developed the Alice microrobot, helped with the software interface and writing of this paper. Jesper Blynel helped with the experiments, project management, and writing of this paper.

[1] They certainly showed to be excellent control systems for biological organisms!

Figure 1: A Khepera robot equipped with a linear camera is positioned in an arena 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.

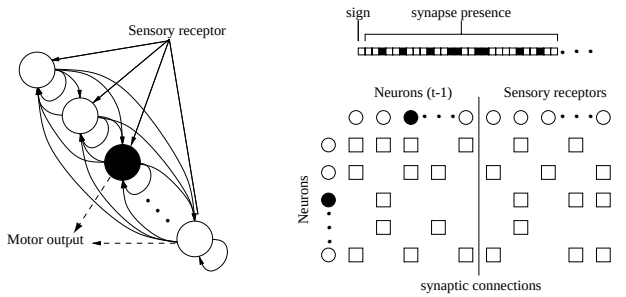

Figure 2: Network architecture (only a few neurons and connections are shown) and genetic representation for one neuron. *Left*: A conventional representation showing the network architecture. *Right*: The same network unfolded in time (neurons as circles, synaptic connections as squares). The neurons on the column receive signals from connected neurons and receptors shown on the top row. The first part of the row includes the same neurons at the previous time step to show the connections among neurons. Sensory neurons do not have interconnections. The signs of the neurons (white = excitatory, black = inhibitory) and their connectivity pattern is encoded in the genetic string and evolved.

been used to drive the wheels of the robot in order to follow lines using a hand-designed program (Indiveri & Verschure 1997). In another line of work, Lewis *et al.* (2000) developed an analog VLSI circuit with four spiking neurons capable of controlling a robotic leg and adapting the motor commands using sensory feedback. This neuromorphic circuit consumes less than 1 microwatt and takes less than 0.4 square millimeters of chip area.

In this paper, we take a different approach to the generation of functional spiking circuits in that *a)* we show that the architecture of functional spiking circuits can be evolved instead of hand-designed and *b)* we describe a software implementation of evolutionary spiking circuits in *digital* microcontrollers with size and power consumption competitive with analog VLSI chips mentioned above. Since the software operates at the level of *single bits* and uses few logic functions, both the neural circuit and evolutionary algorithm fit on the same chip in a few bytes of data memory. The method is tested with a 2-cm long microrobot that must develop navigation abilities without human intervention and external cables.

Evolutionary Spiking Circuits for Robot Navigation

In previous work, some of us (Floreano & Mattiussi 2001) investigated the evolvability of spiking circuit architectures for vision-based navigation of a mobile robot. A Khepera robot with a linear camera was asked to navigate in a rectangular arena with textured walls (figure 1). The walls were filled with black and white vertical stripes of random size.[2] The fitness value was proportional to the amount of forward movement during 40 seconds of robot life. Therefore, robots that could move straight and avoid walls had higher probability of reproduction.

The architecture of a fully recurrent network of 10 spiking neurons connected to 16 spiking visual receptors was genetically encoded and evolved using a standard genetic algorithm (Goldberg 1989) with a population of 60 individuals sequentially evaluated on the same robot. The architecture is genetically represented by a binary string composed of blocks corresponding to each neuron. The first bit of a block encodes the sign of the corresponding neuron (1, -1) and the remaining $n + s$ bits encode the presence/absence (1, 0) of a connection from the n neurons and from the s receptors (figure 2). The synaptic strengths of all existing connections are set to 1. The spiking neuron model used in those experiments included the response profile of synaptic and neuron membranes to incoming spikes, time delays to account for axon length, and membrane recovery profile of the refractory period (Gerstner 1991). The parameter values for the equations were predefined and fixed for all networks. The neural network software, consisting of several hundred lines of C++ code in order to simulate the entire network activation and signal transmission, was run by an external workstation communicating with the robot every 100 ms through a serial connection that

[2]Random size was used to prevent development of trivial solutions whereby the neural network would use the size of the stripes to measure distance from walls and self-motion.

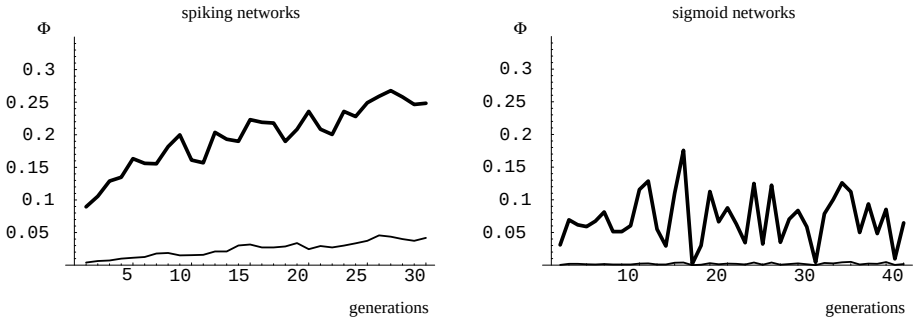

Figure 3: Fitness values obtained on the physical robot Khepera (best fitness = thick line; average fitness = thin line). Each data point is the average of several evolutionary runs with different random initializations. *Left*: Evolution of spiking networks (average over six runs). *Right*: Evolution of connectionist sigmoid networks where 10 additional generations per run were allowed to check for signs of improvement (average over three runs).

transmitted visual information and motor commands. The graph on the left of figure 3 displays population mean and population best fitness values averaged across six runs of evolutionary spiking networks. Fitness values between 0.15 and 0.2 already correspond to robots that can move forward and avoid walls. Further fitness gains correspond to faster and smoother trajectories. As a comparison, we performed another set of experiments with networks of sigmoid neurons using the same architecture, genetic encoding, and evolutionary parameters. The graph on the right of figure 3 shows the average fitness values measured across three runs of sigmoid networks. Despite allowing for an extra ten generations, none of the evolutionary runs could improve fitness values along generations. Occasional higher values are given by individuals that perform wide circles, independently of the sensory input, until they remain stuck against a wall.

Neural and behavioral analysis of evolved spiking controllers (figure 4) indicated that neurons exploit temporal correlation of spiking activity from sensory neurons as well as from internal neurons in order to avoid walls and maintain a smooth forward trajectory (Floreano & Mattiussi 2002).

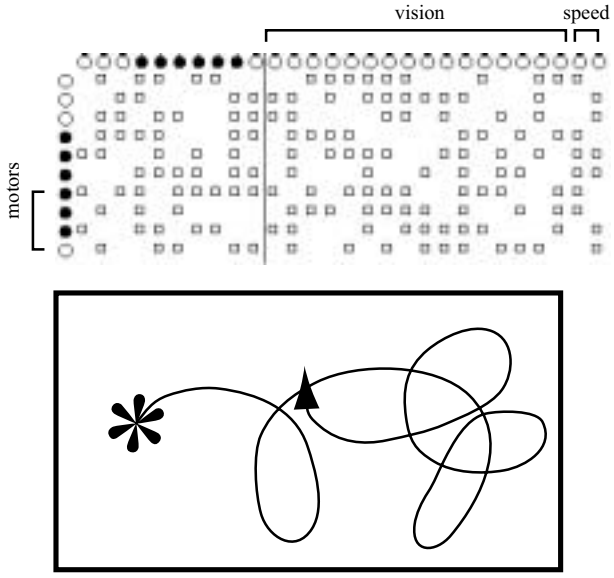

Figure 4: *Top*: Architecture of the best spiking controller after 30 generations. Black circles = inhibitory neurons, white circles = excitatory neurons. *Bottom*: Typical trajectory of robot displayed by this spiking controller. The asterisk indicates the starting point. The curvature of the trajectory depends on the pattern of stripes seen by the robot.

Evolutionary Spiking Circuits in a Microcontroller[3]

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. Microcontrollers are used for a wide range of smart devices, such as microwave ovens, telephones, washing machines, car odometers, and credit cards. More than 3.5 billion microprocessor units are sold each year for embedded control, exceeding by more than an order of magnitude the number of microprocessor units sold for computers (Katzen 2001).

Most applications using microcontrollers require very low power consumption, small size, robustness to hard operating conditions, and low price. These features come

[3]© Dario Floreano, 2001. The copyright applies to the software implementation of the neural circuit, of the evolutionary algorithm, and of their combination. The software is freely available to academic institutions for educational and research purposes. It can be obtained from `[information to be added for camera-ready version]`.

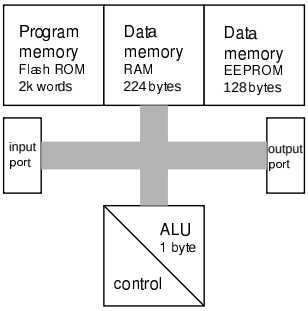

Figure 5: Components of a microcontroller with von Neumann architecture. The microprocessor unit is composed of an Arithmetic Logic Unit and of control devices to move data from/to memory banks and input/output ports. The memory banks are sometimes organized in physically separated locations. For example, a microcontroller may use ROM memory to store a program composed of a maximum of 2k instructions, each 14 bits long; 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. The family of PIC microntrollers used in these experiments use the Hardward architecture whereby the data memory and instruction memories can be accessed in parallel through two different buses.

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 and complex functions are approximated by combination of simpler ones or represented as look-up tables of input-output numbers.

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 precise coding of spiking circuits can be provided by spiking times, rather than by non-linear, real-valued, activation functions used in connectionist neuron models. In other words, a few logic operations (such as AND and NOT) and instructions to move around single bits over time would be sufficient to build in tiny chips large circuits of spiking neurons that display complex abilities and behaviors.

The robotics experiment described in the previous section showed that artificial evolution can easily discover quite powerful spiking circuits by exploring only the space of neuron sign and connectivity. Both variables can be described by a single bit (1 = positive sign, connection enabled; 0 = negative sign, connection disabled)

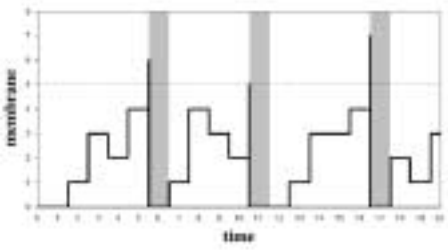

Figure 6: Behavior of a neuron with constant firing threshold.

and therefore can be efficiently stored and easily manipulated in microcontrollers.

The next two subsections will describe the neuron and evolutionary model, respectively, as well as the implementation idea. The section that follows will describe an example of this implementation where a microrobot equipped with a microcontroller evolves without human intervention and external computers in less than two hours the ability to move around a maze.

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

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 precision and relatively high computing speed. Therefore, the neuron model used here is a simple integrate-and-fire model with leakage and refractory period.

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

1. *Refractory period.* If the neuron has emitted a spike within the previous Δt, do not update its membrane potential. In these experiments, $\Delta t = 1$.

2. Compute the *contribution of incoming spikes* e_i^t as the sum of spikes o_j^t at time t through existing connections w_{ij} weighted by the sign of emitting neurons s_j:

$$e_i^t = \sum_j^N o_j^t w_{ij} s_j \tag{1}$$

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

3. *Update membrane potential* v_i^t by adding the contribution of incoming spikes to the available potential,

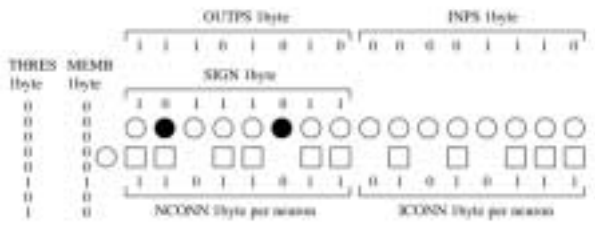

Figure 7: Digital representation of one neuron in the microcontroller.

but do not go below a minimum level v_i^{min}

$$v_i^t = \begin{cases} v_i^{t-1} + e_i^t & : \quad v_i^{t-1} + e_i^t \geq v_i^{min} \\ v_i^{min} & : \quad \text{otherwise} \end{cases} \tag{2}$$

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

4. *Spike generation.* If the membrane potential is larger or equal to a maximum level v_i^{max}, set the neuron output to 1 (spike) and the membrane potential to its minimum value v_i^{min}; otherwise set the neuron output to 0 (no spike) and do not modify the membrane potential.

$$o_i^t = \begin{cases} 1 \text{ and } v_i^t = v_i^{min} & : \quad v_i^t > v_i^{max} + r^t \\ 0 & : \quad \text{otherwise} \end{cases} \tag{3}$$

where here $v_i^{max} = 5 \ \forall i$ 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.* Subtract a leaking constant k_i from the membrane potential, but do not go below the minimum level v_i^{min}

$$v_i^t = \begin{cases} v_i^t - k_i & : \quad v_i^t - k_i \geq v_i^{min} \\ v_i^{min} & : \quad \text{otherwise} \end{cases} \tag{4}$$

Here $k_i = 1 \ \forall i$.

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 2. The sign of the neuron determines the effect of its spikes on other neurons (equation refeq:prespikes). The presence of a spike in the sensory neuron is determined by the activity of sensors.

Since the Arithmetic Logic Unit (figure 5) of the microcontroller used in these experiments can operate on 8 bits in parallel, the circuit described here is composed of 8 interconnected neurons and 8 sensory neurons, which are stored in the RAM data memory (figure 7). The output (1 = spike; 0 = no spike) of the 8 neurons is stored in byte OUTPS (1 bit per neuron) and that of the 8 sensors in byte INPS (1 bit per sensor). The sign of the 8 neurons is stored in byte SIGN (1 = positive; 0 = negative). The connectivity pattern of one neuron is stored in one byte of block NCONN (connections from neurons) and in one byte of block ICONN (connections from sensors). NCONN and ICONN are blocks of 8 bytes each. The membrane potential of one neuron is stored in one byte of block MEMB, which is composed of 8 bytes too. The maximum membrane potential is constant for all neurons and the stored in byte THRES; the minimum membrane potential is 0 for all neurons and thus does not require memory storage. The entire spiking circuits takes 20 bytes (INPS, OUTPS, SIGN, THRES, 8 x MEMB, 8 x NCONN, 8 x NCONN). Nine additional bytes are used to store random numbers, counters, and temporary variables (which are shared with the evolutionary algorithm described in the next section).

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 the number of active bits that result from the `AND` function of byte INPS and ICONN. Continue with spikes from positive neurons: increment MEMB variable by counting the number of active bits that result from the `AND` function of OUTPS and 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 of SIGN and NCONN. The decrement is stopped before MEMB goes below zero (which is readily signalled by a bit flag in a housekeeping byte of the microcontroller).

3. *Spike generation.* Compute random value for k_i and check whether MEMB is equal or larger to THRES incremented/decreased by k_i. If so (spike), set the corresponding bit in OUTPS to 1 and MEMB to zero. Otherwise (no spike), set corresponding bit in OUTPS to 0.

4. *Leakage.* If MEMB is greater or equal than leaking constant (1), decrement it by constant value.

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 up a neuron at random and change 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 4MHz, the entire network is updated in approximately 2 ms. In some case, such as for the robotics experiment described here, the entire network can be updated faster than the time required by the physical sensors to collect information. Therefore, inbetween new sensory values, INPS is set to all 0's while the neurons continue to be updated using only internally generated spikes. Alternatively, one could decide to update INPS, or even the entire network, only when new sensory inputs are available.

Evolution Model and Implementation

The experiments on vision-based navigation described in section suggested that functional spiking circuits can be evolved by genetically encoding only the sign of the neurons and the presence/absence of synaptic connections (see figure 2). The same genetic encoding has been used for the neuron model used here. Consequently, the genetic string of the spiking circuit described here 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).

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 (Whitley & Kauth 1988; Syswerda 1989), has been chosen. The implementation used here, designed to maximize exploration while preserving the best solution obtained so far, works as follows:

1. Randomly generate a population of binary strings and initialize their fitness values to zero.
2. Pick an individual at random, 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 over the old one, 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" (Kimura 1983) on the genetic landscape. This may be a useful property for evolution of small converged populations (Harvey & Thompson 1996).

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.

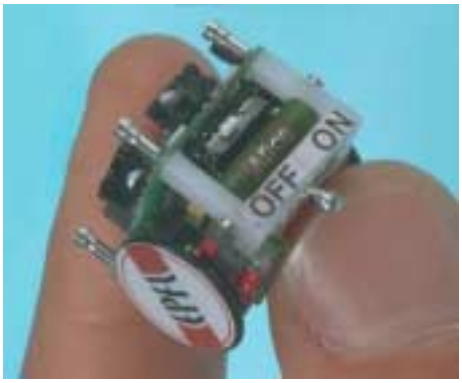

Figure 8: The mobile micro-robot Alice in the basic version.

It is useful to store the population 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 (up to 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 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 (step 1 in the procedure described above), 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.[4] In the experiments described below, we have decided to keep a copy of the entire population in the free space of the RAM memory, use it for evolution, and copy it to the EEPROM at predefined large intervals.

The Alice Microrobot

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 (Caprari *et al.* 1998). Its long energetic autonomy between 2 (regular batteries) and 10 hours (with an

[4]These errors represent a free mutation operator during reading, if the corresponding error-checking routine is not used, and could substitute the XOR and random instructions mentioned above.

Dimensions	21 x 21 x 22 mm
Weight	5 g
Velocity	40 mm/s
Power consumption	4 mW - 10 mW
System autonomy	up to 10 hours
Mechanical structure	plastic frame and PCB
Motors	2 bi-directional Swatch motors
Motion	2 wheels on the minute axis
CPU	PIC16F628 @ 4 MHz
Energy source	3 button batteries V377 (3 x 1.5V, 23 mAh)
Extra battery pack	rechargeable NiMH battery 3/V40H (3 x 1.2V, 40 mAh)
Sensors	4 infrared proximity sensors
Communication	tiny cables (debugging/analysis)

Table 1: Features and components of the Alice robot.

Flash ROM	2 Kwords (14 bits)
RAM data	224 Bytes
EEPROM data	128 Bytes
Pins	18, (15 I/O)
Power consumption	¡ 2.0 mA @ 5.0 V, 4.0 MHz

Table 2: Data of the PIC16F628 microcontroller.

extra battery module) make it unique in its class. Alice is a programmable and modular robot. In its basic configuration it has 2 motors for locomotion, 4 active infrared sensors, a microcontroller and 3 button batteries. A number of modules can be added on its top, such as vision, radio, and an extra battery pack. Table details all the parts and characteristics of the version used for these experiments. Thanks to its programmability and interface with personal computers, Alice has been used in various research (Caprari, Arras, & Siegwart 2001; Siegwart *et al.* 1998) and educational projects (Caprari, Arras, & Siegwart 2000).

The microcontroller PIC16F628 from Microchip is the central part of the electronics and control of the robot. It directly drives the 2 low-power, step motors through 6 pins and serially reads the values of the analog-digital converter for sensor measurement. The PIC16F628 is a RISC (Reduced Instruction Set Computer) microcontroller whose entire package occupies approximately 2 mm^2 in the version used here. Most of the memory resources are taken by the management of the sensors and motors. The software core is composed of a very simple real time operating system which handles 5 different time-critical tasks (right and left motors, communication, sensor reading and switching) that have the priority of the spiking circuit update. The infrared sensors have a limited range of 2 to 3 cm, which is similar to the

Figure 9: The mobile micro-robot Alice with the extra battery module providing up to 10 hour of energetic autonomy.

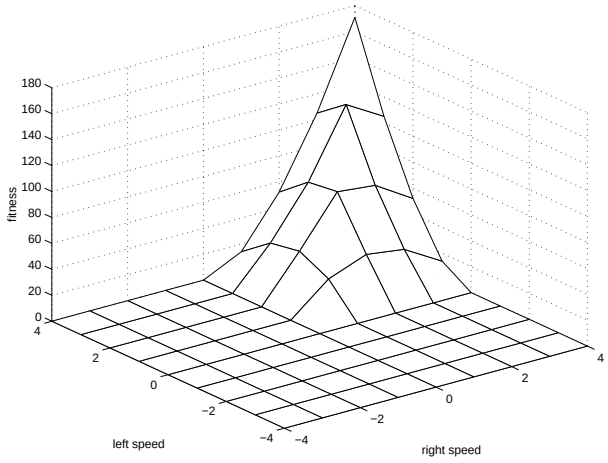

Figure 10: Fitness values shown as a function of wheel speeds (sensor activation $i = 0$). Positive values indicate forward rotation.

size of the robot itself. The last bit of a sensor activation that is quite random is read every 50 ms to re-initialize the pseudo-random number generator required to update the spiking circuit and run the evolutionary algorithm. For the repeated experiments described in this paper, a rechargeable battery-pack module (figure 9) has been developed instead of replacing every two-three hours the button batteries.

Evolutionary Experiments

The robot was positioned in a 25 by 18 cm arena with a wall in the middle and asked to move forward without hitting the walls. The fitness was computed and accumulated at each sensory-motor cycle using a truncated version of the often-used function to evolve straight navigation and obstacle avoidance (Floreano & Mondada 1994)

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

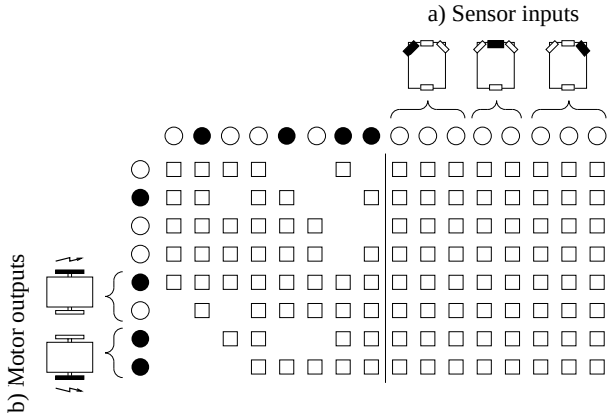

Figure 11: Architecture and connectivity of a randomly generated network. Squares indicate presence of a synaptic connections. Only the connections among neurons and neuron signs are genetically encoded and evolved. All neurons are always connected to all sensors (see text). The small drawings illustrate how the network is connected to the sensors (a) and motors (b) of the robot.

where V is the sum of the speeds of the two wheels, ΔV is the absolute difference between the two wheel-speeds and i is the activity of the most active sensor. Since the Alice robot does not have wheel encoders to measure wheel rotation, the speed values used in the formula are taken from the output of the neural circuit. In addition, whenever one of the wheel speeds was in backward rotation, the entire fitness was set to 0. Finally, in order to use as few bytes of memory as possible to store the fitness value, each of the three terms were scaled so the maximal fitness value of each sensory-motor step multiplied by the total number of steps could fit in a single byte. Figure 10 plots the actual fitness values as a function of wheel speeds, provided that all infrared sensors are inactive.

The spiking circuit was composed of 8 neurons and 8 sensory units, as described above. Only the three frontal proximity sensors were used. To compensate for the steep drop-off response profile, the activation of each sensor is scaled in the range $[0, 7]$ and coded on three bits by setting active bits proportionally to the sensor activation, as shown in table . The network has a total of 8 sensor input bits. 3 bits are used for the front left and the front right sensor each and 2 bits for the center front sensor. In other words, the more active the sensors are, the more sensory units emit a spike. Furthermore, given the simplicity of the navigation task considered here, we decided to genetically encode and evolve only the signs of the neurons and the connections among neurons, leaving all neurons fully connected to all sensors. That means that each neuron has the same sensory information and cannot tell where the walls are. Although

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

Table 3: Coding of the sensory inputs for the three frontal sensors. The network has 8 sensory input bits. For the left and right frontal sensor all three bits are used while for the central front sensor the last bit is discarded.

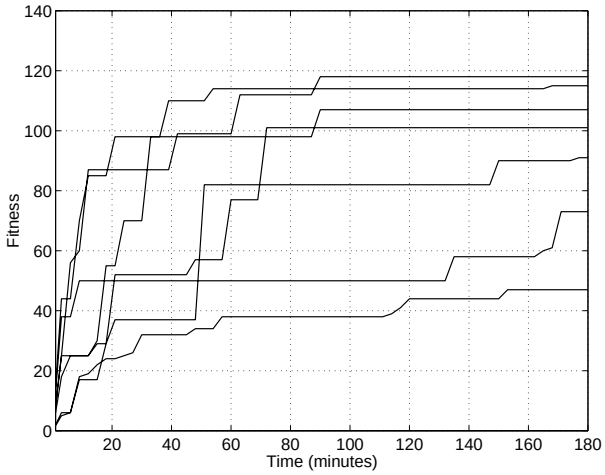

Figure 12: Fitness of the best individuals in 7 evolutionary runs. Data points are sampled every 3 minutes.

this may slightly complicate the discovery of a good navigation stategy, it results in very short genetic encodings (1 byte for SIGN and 8 bytes for the NCONN block). The sensory input was computed anew every 28 ms during which the network was let free to update its entire state several times (once every 2ms for this microcontroller run at 4 MHz). At the end of the 28 ms, the velocities of the two wheels were set proportionally to the number of spikes of the motor neurons. The speed of each wheel was the algebraic difference of the spike count of two motor neurons, one moving it forward and the other moving it backward. The result was scaled in the range $[-4, 4]$.

For each experiment, a population of 6 individuals was randomly initialized and evolved for 3 hours using onboard batteries. Each individual was tested for 10 seconds while a random movement for 3 seconds was applied between individuals. 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. Figure 12 plots the fitness values of the best individuals for each of the 7 experiments with different random initializations of the population.

Figure 13 shows the path traced over 10 seconds by

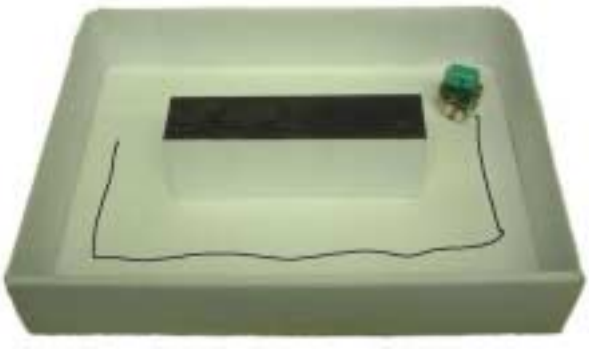

Figure 13: The robot in the environment used in the experiments. The environment is made out of white cardboard and has the dimensions 25 x 18 cm with an 12 x 3 cm obstacle in the middle. The path covered by an evolved robot in10 seconds is over-layed from a video footage.

one of the best evolved robots with a fitness of approximately 120. Evolved robots go forward and whenever sensor activity becomes large they turn on the same side. The degree of rotation and speed depends on the sensory activity level and is quite tuned to the geometry of the environment. In best robots, this strategy corresponds to a careful wall following strategy. Small occasional jigs are most likely caused by the noise applied to the firing threshold. Robots with lower fitness values tend to rotate more frequently and thus remain in the same area of the maze, but always manage to move away from the walls.

Discussion

There is nowadays a consensus in Evolutionary Robotics (Harvey *et al.* 1997; Lipson & Pollack 2000; Nolfi & Floreano 2001) that more complex genotype-to-phenotype mappings are necessary to scale up to the evolution of more complex (not necessarily complicated) artificial organisms. At the same time, it is necessary to search for the type of neural bricks that are most suitable for interface with the real world, recombination, and mutation. While most researchers use continuous-time recurrent neural networks, we believe that the spiking circuits described here represent a very promising alternative because not only they can display similarly complex non-linear time-dependent dynamics, but also are easily mapped into low level digital circuitery (which is more widely available and usable than analog VLSI). In addition, our preliminary experiments indicate that they may be more suitable for evolutionary systems in the real world.

The speed and reliability of the results obtained with the micro-robot Alice are also a promising indication of the efficacy of the models and methods suggested here to evolve circuits of spiking neurons in digital microcontrollers with a few bytes of memory. The experiments obtained with the vision-based Khepera robot let us think that the approach will scale up to more complex tasks and sensory inputs. Provided that enough data RAM is available, it is straightforward to implement circuits where the number of neurons and/or sensors is a multiple of 8. In addition, it is possible to build a network of several microcontrollers, each implementing a spiking module, and let them communicate spikes in parallel through input/output ports or using a time-stamp communication protocol, such as the Address-Event Representation (Lazzaro *et al.* 1993). Obviously, that would require a different type of genetic encoding, possibly one based on gene expression and module growth.

The approach described here could also find its way in a large set of intelligent devices with embedded microcontrollers. The microcontroller described here is an instance of a much larger variety of devices where the method can be adapted to meet different memory or energetic constraints. For example, to stay with the PIC family used here, the program could be easily modified to drastically improve instrcution memory over data memory, or viceversa. Similarly, energy consumption could be significantly reduced by putting the microcontroller in standby mode (power consumption drops to less than $1\mu A$) between sensory updates instead of continuously updating the neural network. For several applications, such as adaptive dishwashers and hairdryers, smart credit and identity cards, etc. the requirements for adaptation, non-linear dynamics, and input/output processing are likely to be less demanding than for autonomous mobile robots. In that case, much simpler spiking circuits could be evolved in less digital space.

Future work

Our current work is aimed at characterizing the dynamics and evolvability of the models described in this paper. We are also expanding the model to include vision inputs and distributed implementation across different microcontrollers within the same robot. The approach is being applied in both wheeled microrobots and flying robots where weight, size, and energetic constraints are a major issue.

In parallel, we are exploring new types of genetic encodings for evolution of adaptive and self-repairing circuits of spiking neurons implemented directly in reconfigurable digital hardware.

Acknowledgements

This work was partially supported by Swiss NSF grant No. 620-58049. Dario Floreano is grateful to Jean-Daniel Nicoud for providing the *PICgenial* board (distributed by Didel SA, www.didel.com) where the evolutionary spiking circuit was initially developed and tested.

References

Arbib, M. A., ed. 1998. *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA: MIT Press-Bradford Books.

Bower, J. M., and Beeman, D. 1994. *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*. New York: Springer Verlag.

Caprari, G.; Arras, K. O.; and Siegwart, R. 2000. The autonomous miniature robot alice: From prototypes to applications. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)*. IEEE Press. 793–798.

Caprari, G.; Arras, K. O.; and Siegwart, R. 2001. Robot navigation in centimeter range labyrinths. In Rückert, U.; Sitte, J.; and Witkowski, U., eds., *Proceedings of the 5th International Heinz Nixdorf Symposium: Autonomous Minirobots for Research and Edutainment, AMiRE 2001. Paderborn, Germany*. HNI-Verlagsschriftenreihe. 83–92.

Caprari, G.; Balmer, P.; Piguet, R.; and Siegwart, R. 1998. The autonomous micro robot alice: A platform for scientific and commercial application. In *Proceedings of the 9th International Symposium on Micromechatronics and Human Science*. 231–235.

de Garis, H. 1996. CAM-BRAIN. the evolutionary engineering of a billion neuron artificial brain by 2001 which grows/evolves at electronic speed inside a cellular automata machine (CAM). In Sanchez, E., and Tomassini, M., eds., *Towards Evolvable Hardware. The Evolutionary Engineering Approach*. Berlin: Springer Verlag. 76–98.

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

Floreano, D., and Mattiussi, C. 2002. Evolution and Analysis of Spiking Neural Circuits for Autonomous Robots. submitted.

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

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

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

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

Harvey, I.; Husbands, P.; Cliff, D.; Thompson, A.; and Jakobi, N. 1997. Evolutionary Robotics: The Sussex Approach. *Robotics and Autonomous Systems* 20:205–224.

Horiuchi, T. 2001. Neuromorphic VLSI links. Web page: www.enee.umd.edu/class/enee719t/links.html.

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

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

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

Katzen, S. 2001. *The Quintessential PIC Microcontroller*. London: Springer Verlag.

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

Koenig, P.; Engel, A. K.; and Singer, W. 1996. Integrator or coincidence detector? The role of cortical neuron revisited. *Trends in Neuroscience* 19:130–137.

Lazzaro, J.; Wawrzynek, J.; Mahowald, M.; Sivilotti, M.; and Gillespie, D. 1993. Silicon auditory processors as computer peripherals. *IEEE Transactions On Neural Networks* 4(3):523–528.

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

Lipson, H., and Pollack, J. B. 2000. Automatic design and manufacture of robotic lifeforms. *Nature* 406:974–978.

Maas, W., and Bishop, C. M., eds. 1999. *Pulsed Neural Networks*. Cambridge, MA: MIT Press.

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

Nolfi, S., and Floreano, D. 2001. *Evolutionary Robotics: Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA: MIT Press. 2nd print.

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

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

Siegwart, R.; Wannaz, C.; Garcia, P.; and Blank, R. 1998. Guiding mobile robots through the web. In *Workshop Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, Victoria, Canada*.

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

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