Generation of Non-Reactive Behaviors in Robotic Agents by means of Evolutionary Algorithms

J. Catacora, Universidad Nacional de Ingenieria, 2012

ABSTRACT

This work focuses on the generation of non-reactive behaviors, and particularly of working memory behaviors, in robotic agents through a neuroevolutionary approach. The motivation of the author was to assess the competence of evolutionary algorithms in combination with artificial neural networks to develop a working memory and to discover the underlying mechanisms that engender this ability in intelligent agents. The experimental setting involved the T-maze alternation task, which requires a robotic agent to remember the location of the goal of the maze. Furthermore, the competence of nine neural architectures to promote the evolution of a working memory was tested by using a genetic algorithm to tune their variable parameters with the aim of achieving agents able to solve the proposed task. The experimental results showed that only 4 out of 9 architectures favored the evolution of a working memory, while the remaining neural models showed no signs of evolving this ability. The best evolved agents were further tested in modified environments and for varying simulation times. The results of these additional experiments revealed that memory-endowed agents rely on sensing the time and synchronizing their movements accordingly, i.e. a temporal working memory. In addition, the analysis of the sensory and neural activity recorded from a given agent while it was performing the task determined that its ability to sense the passage of time arises from at least one time-dependent neural signal (for instance, a neuron output that grows linearly throughout the simulation) produced internally by its neurocontroller, which could serve as a timer. Upon further experimentation, it was discovered that the emergence during an evolutionary process of neurocontrollers able to generate these time-synchronized signals is directly linked to the internal dynamics, namely the activation functions and network topologies, of each neural architecture being used, and more specifically to the probabilistic likelihood of producing those signals from a random set of the variable parameters associated with each model, such as synaptic weights and time constants, among others. Ultimately, this last finding provides the rationale for why in the initial experimental setting some neural architectures were more advantageous than others to the evolution of a working memory.

1. Introduction

In recent times modern Robotics has moved from the thoroughly controlled industrial environment to the chaotic natural environment where humans live. In these early experiences, robots have carried out relatively simple tasks in the fields of service and entertainment; and in the future we can only expect an ever-growing interaction between humans and robots.

However, the natural environment in which we live is typically unstructured and constantly changing; therefore, as robots are confronted to more natural settings, to hand-design them based on control policies derived from our own perceptions and intuitions would be an increasingly herculean task, since such approach would require to account for every possible reconfiguration of the environment, including every interaction with other autonomous agents.

In light of this limitation, over the past twenty years robotics engineers have taken inspiration from biology in order to create more robust and adaptable systems. It is easy to understand the benefits of this new paradigm just by looking at how perfectly adapted to their environments all living beings on Earth are. Mammals, birds, ants, even the cells of our immune system, all these organisms display behaviors desirable to be replicated in new generations of robots.

Researchers also noted that some of these behaviors, in addition to being intrinsic to each species and particular to the environments in which they have developed, are far too complex to have been acquired ontogenetically during the lifetime of an individual; instead, it is necessary to conceive their emergence through a lengthy evolutionary process over the course of many generations. It was from this basic idea that the field of Evolutionary Robotics was born; a field in

which scientists attempt to mimic evolutionary mechanisms in order to develop robots with greater flexibility, autonomy, adaptability and robustness; well-suited to handle the ever-changing real world.

So far the major accomplishments of Evolutionary Robotics have been in the areas of locomotion of legged robots, autonomous navigation of wheeled or flying robots and swarm robotics [1]. Despite this great progress, the behaviors obtained, although robust and adaptive, are still relatively simple and mostly reactive and the achievement of more sophisticated non-reactive behaviors, especially of higher level cognitive processes such as learning, understanding, communicating, applying, analyzing, creating, remembering or planning, remains a grand challenge within the field [2],[3]. A non-reactive behavior, also known as non-deterministic or deliberate behavior, is defined as one in which the action is executed independently of any immediate sensory information; thus, in order to achieve this, an intelligent agent must be able to make decisions based on its own internal cognitive dynamics [4],[XXX].

The focus of this study is to investigate the feasibility of evolving a working memory, i.e. the ability to remember an object, stimulus, or location that is used within a testing session, but not typically between sessions [XXX], in robotic controllers. A working memory behavior is a kind of non-reactive behavior, because an intelligent agent, aided by its recollection of past states, is required to perform different actions under the same immediate sensory information. Moreover, a working memory, along with the many other memory skills that humans and animals possess, is a crucial stepping stone towards achieving the more sophisticated higher level cognitive processes mentioned previously [XXX].

The development of a working memory is a particularly difficult task in the field of Evolutionary Robotics. Not every structure that could be regarded as the controller or 'brain' of an intelligent agent, such a neural network, is necessarily capable of evolving this ability. Said structures must exhibit special characteristics that foster the emergence of complex internal dynamics. Nevertheless, it is not straightforward to determine a priori which structures will or will not yield good results; hence, the best way to proceed is to test these structures in controlled experiments [5]. This difficulty was taken in this study as a perfect opportunity to also research the competence of different structures to engender a working memory.

To address the two research goals mentioned above, the experimental setting of this study involves the T-maze alternation task, in which a simulated robotic agent is given two trials to reach the goal of the maze, which in turn is switched from arm to arm between epochs. This is an alternation memory task, and as such it is expected that it will favor agents that display a working memory behavior. Furthermore, artificial evolution is applied only to the controllers of the robotic agents being tested in the proposed task. Nine neural architectures are used as the basis for the evolving controllers, and their competence to engender a working memory is tested by using a genetic algorithm to tune their variable parameters, such as synaptic weights, time constants, etc., with the aim of achieving agents able to solve the T-maze alternation task.

2. Methods

The present work has been implemented entirely in a virtual setting. Hence, the evolved agents were not tested in the real world. Taking this restraint into account, some simplifications were made to the physical simulation, such as the absence of the robot dynamics, the sharing of the same sensor or motor lookup table by several devices, among others. Nevertheless, an emphasis was indeed placed to the addition of environmental noise in order to promote the evolution of robust agents over fortuitous solutions [6],[7].

It is also worth mentioning that the entire computational framework, comprising the physical simulation of the environment and the robotic agent, as well as the genetic algorithms and operators, was programmed in Labview 8.6.

2.1. Task Environment

The task environment is a T-maze, whose goal zone is switched from left to right and vice versa every epoch (Figure 1). The dimensions of the maze were reduced to a minimum in order to shorten the running time of the evolutionary

algorithm. As a result, the T-maze is composed of a total of five 10 cm by 10 cm squares. Furthermore, the T-maze has colored start and goal squares, denoted by colors gray and black respectively. Meanwhile, the remaining squares are white-colored.

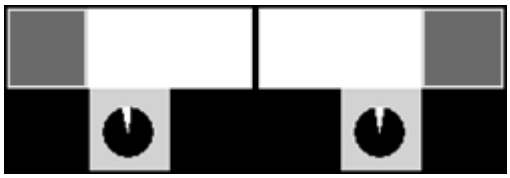

Figure 1. T-Maze.

2.2. Robot Simulation

2.2.1. Robotic Agent

For this experience a differential drive mobile robot was simulated. The agent has 2 wheels of 5 cm in diameter each; the distance between the centers of both wheels is 5.5 cm; and the contour of the robot is a circle 6 cm in diameter, whose center lies precisely at the midpoint of the segment joining the centers of the two wheels.

The automaton possesses two continuous rotation servomotors, whose open-loop speed is controlled by a PWM signal (Pulse-Width Modulation), as shown in Figure 2. These actuators reach a top speed of 50 rpm, which equates to a maximum linear speed of 12.3 cm/s.

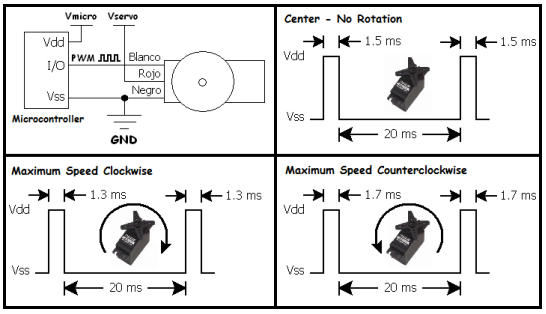

Figure 2. PWM Control Signal.

The agent employs five infrared (IR) reflective sensors to detect obstacles located at distances between 2 mm and 42 mm. These sensors are arranged at -75°, -37.5°, 0°, 37.5° and 75° relative to an axis perpendicular to the line connecting the centers of the two wheels and parallel to the ground. In addition, these distance sensors are placed 2 mm within the peripheral circumference of the mobile robot; such restriction limits the operation of these devices to their monotonic sensing range (Figure 3).

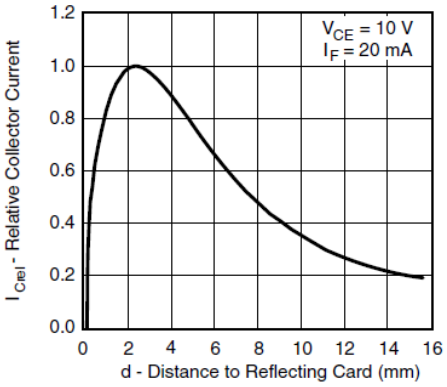

Figure 3. Relative Reverse Light Current vs. Distance.

The robot also receives information of its surroundings from four color sensors installed on its bottom surface, 0.5 cm above ground level. The output signals of these four devices are combined logically so that ultimately only a single ternary signal, indicating that the color of the floor is either white, gray or black, enters the controller.

The following figures show the dimensions and components of the robotic agent:

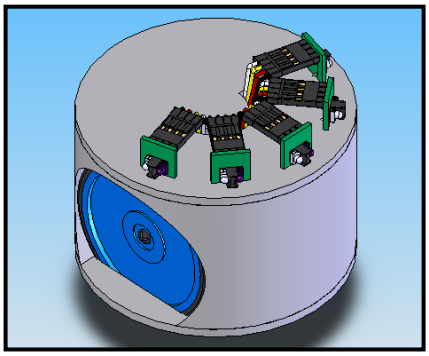

Figure 4. Isometric view of the robot.

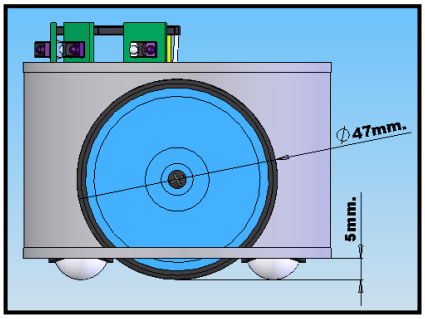

Figure 5. Front view of the robot.

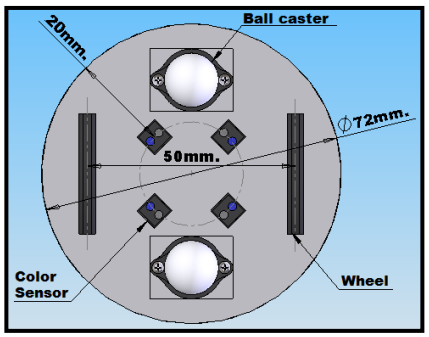

Figure 6. Bottom view of the robot.

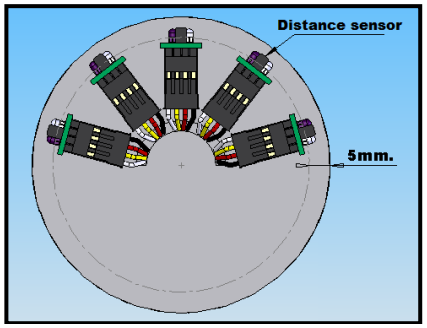

Figure 7. Top view of the robot.

2.2.2. Kinematics

The motion of the automaton is simulated by means of a kinematic model for a differential drive vehicle. As the differential drive configuration allows varying the speed of each wheel individually, the instantaneous center of curvature (ICC) of the vehicle is located on the common axis of both wheels. Therefore, the instantaneous rotational speed ω of the vehicle around the ICC and the signed distance R from the ICC to the midpoint between the wheels are described by the following equations:

$$\omega = \frac{V_R - V_L}{L}; R = \frac{L}{2} \cdot \frac{V_R + V_L}{V_R - V_L}$$

Where V_R and V_L are the tangential velocities of the right and left wheels respectively, whereas L corresponds to the distance between the two wheels. According to [8], given the current position (x, y) and orientation ϑ of the robot at time t, its position and orientation (x', y', ϑ') at the next iteration $t + \delta t$ are solved as follows:

$$ICC = \left[x - R \cdot \text{sen}(\theta), y + R \cdot \cos(\theta)\right]$$

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega \cdot \delta t) & -\text{sen}(\omega \cdot \delta t) & 0 \\ \text{sen}(\omega \cdot \delta t) & \cos(\omega \cdot \delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega \cdot \delta t \end{bmatrix}$$

The time step δt used to simulate the motion of the agent is 0.1 seconds at all times.

As the automaton moves through the T-maze, it could at any time collide with neighboring walls. If a collision occurs, the agent will be repositioned inside the maze at the next simulation step. This repositioning is not governed by any set of collision equations; instead the robot is simply placed in the center of the square where the collision occurred and its orientation prior to the collision is maintained. This method is deemed appropriate, because it is expected that the evolved agents never collide with the walls of the maze.

2.2.3. Servomotors

The simulation of the two servomotors is based on a single lookup table (Table 1) developed from the theoretical behavior of real actuators. Specifically, the input-output relationship of a particular model of servomotor (PARALLAX 900-00008) was used to create the aforementioned lookup table. The datasheet of this component indicates that for a PWM range between 1300µs and 1700µs the motor speed varies proportionally between 50 rpm (clockwise) and 50 rpm (counterclockwise). Within the simulation, the controller outputs, ranges [0, 1] or [-1, 1], are first linearly scaled to the PWM range [1300µs, 1700µs]; then, the resulting pulse widths are used as inputs to the motor lookup table (if necessary a linear interpolation is performed). Finally, the values extracted from the lookup table are regarded as the rotational speed of each servomotor.

Pulse Length [µs]	Motor Speed [rpm]
1300µs	60 CCW
1320µs	54 CCW
1340µs	48 CCW
1360µs	42 CCW
1380µs	36 CCW
1400µs	30 CCW
1420µs	24 CCW
1440µs	18 CCW
1460µs	12 CCW
1480µs	6 CCW
1500µs	0
1520µs	6 CW
1540µs	12 CW
1560µs	18 CW

Pulse Length [µs]	Motor Speed [rpm]
1580µs	24 CW
1600µs	30 CW
1620µs	36 CW
1640µs	42 CW
1660µs	48 CW
1680µs	54 CW
1700µs	60 CW

Table 1. Motor lookup table.

2.2.4. Distance Sensors

The simulation of these devices followed a similar approach as the simulation of the servomotors. One normalized lookup table (Table 2) was created comprising information of two characteristic curves found in the datasheet of a known sensor (TNCD5000): Relative Reverse Light Current vs. Distance and Relative Radiant Intensity vs. Angular Displacement (Figure 3 and Figure 8). This double-entry table returns a sensory output in the range of 0 and 1 based on two input parameters: 1) the distance from the sensor to a given wall and 2) the angle between the sensor beam and the same wall.

Relative Reverse Light Current [mA]									
mm	Degrees								
	-30°	-23°	-15°	-8°	0°	8°	15°	23°	30°
2	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95	0.95
7	0.58	0.66	0.69	0.72	0.68	0.72	0.71	0.68	0.63
12	0.40	0.44	0.49	0.50	0.45	0.50	0.48	0.47	0.42
17	0.19	0.22	0.26	0.27	0.22	0.27	0.26	0.25	0.21
22	0.13	0.15	0.18	0.19	0.15	0.18	0.19	0.19	0.15
27	0.08	0.10	0.14	0.15	0.11	0.14	0.15	0.13	0.12
32	0.06	0.07	0.10	0.10	0.08	0.10	0.10	0.10	0.09
37	0.04	0.05	0.08	0.09	0.06	0.08	0.07	0.07	0.06
42	0.03	0.04	0.06	0.07	0.04	0.06	0.05	0.04	0.03

Table 2. Sensor lookup table.

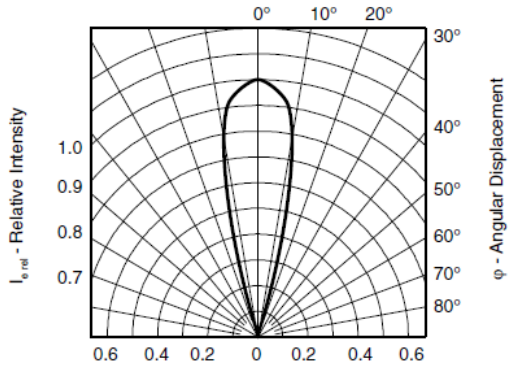

Figure 8. Relative Intensity vs. Angular Displacement.

Within the simulation, the absolute position and orientation of each distance sensor are first estimated thanks to the prior knowledge of the absolute position and orientation of the robot and the relative position of each sensor upon the automaton. Afterwards, since the shape and dimensions of the T-maze are also known, the distance from each sensor to the nearest wall and the angle between them are calculated by means of a geometric analysis. Lastly, these five pairs of distances and angles are entered in the sensor lookup table in order to obtain the simulated sensory readings that will be sent to the controller.

2.2.5. Color Sensors

The simulation of the four color sensors considers that the detection threshold of two of them (CS1 and CS3, located diametrically opposite to each other) was calibrated such that a white surface is read as a logical value of FALSE by these sensors, while a gray or black surface is read as a logical value of TRUE. As for the other two sensors (CS2 and CS4), an output of FALSE corresponds to a white or gray surface, whereas an output of TRUE is associated with a black surface.

Similarly to the procedure employed in the case of the distance sensors, at first the software calculates the absolute position of each color sensor. Then, the logic level of each individual sensor is determined by comparing the previously computed positions to the known locations of the start and goal squares in the T-maze. Next, the resulting array of four signals is transformed by a Boolean function into a single ternary signal (0-white, 0.5-gray, 1-black), which is the one that eventually enters the controller. This function works as follow: if all four sensors have a reading of TRUE, the surface is black; if CS1 and CS3 have a reading of TRUE while CS2 and CS4 have a reading of FALSE, the surface is gray; otherwise, the surface is white.

2.2.6. Noise

Uniform white noise was added throughout the simulation in order to evolve robust agents that are able to cope with changing environmental conditions [6],[7].

Noise of ±5 mm is added to the initial position of the agent (x_0, y_0), while its initial orientation ϑ_0 is

varied between -5° and +5°. Furthermore, ±5% noise is added to the linear and angular displacements experienced by the robot at each time step. With regard to the distance sensors, noises of ±2 mm and ±1.6° are added respectively to the distances and angles that serve as inputs to the sensor lookup table. In the case of the color sensors, noise proportional to the proximity of these devices to a band of color-change is added to their readings by means of a symmetric triangular distribution, which has a maximum probability of 0.5 of switching the logic level of the output of a color sensor when the device is immediately above the line that divides two squares of different floor colors and a minimum switching probability of 0.02 when the device is at a distance greater than or equal to 2 mm from said line. Lastly, if a collision occurs between the agent and its surroundings, noises of ±10 mm and ±22.5° are added respectively to the repositioning location *(x, y)* and orientation ϑ of the robot after the collision.

2.3. Neural Architectures

In this work, a total of 9 neural architectures are used as the basis for the evolving controllers. These structures are: Recurrent Single Layer Perceptron (SLP), Recurrent Multilayer Perceptron (MLP), Recurrent Radial Basis Function Network (RBF) Fully Connected Recurrent Neural Network (FRNN), Continuous-Time Recurrent Neural Network (CTRNN), Echo State Network without leak rate (ESN), Echo State Network with leak rate (ESN+α), Spiking Neural Network (SNN) and Spiking Neural Network with Spike-Timing Dependent Plasticity (SNN+STDP).

All sensory inputs to the neural networks are normalized to lie between 0 and 1. It should be noticed that in the case of the Spiking Neural Networks the initially normalized inputs are subsequently transformed into spike sequences before entering the neurocontroller.

2.3.1. Recurrent Single Layer Perceptron

This network has 2 neurons with sigmoid activation function in the output layer, each of which receives 9 inputs, from: 5 distance sensors, 1 color signal, 2 motor feedbacks and 1 bias. Both outputs are afterwards linearly scaled from the range [0, 1] to the interval [1300μs, 1700μs]; this simulates the generation of the PWM signals that would control the two servomotors. All 18 synaptic weights of the network are confined to the range [-5, 5].

2.3.2. Recurrent Multilayer Perceptron

This network displays a similar configuration as that of the Recurrent Single Layer Perceptron network, except that it additionally possesses a hidden layer of 5 neurons. All 7 neurons have a sigmoid activation function, and all of them receive the same 9 inputs as the neurons of the SLP network. The 57 synaptic weights are bounded between -5 and 5.

2.3.3. Recurrent Radial Basis Function Network

This RBF network is composed of a 5-neuron hidden layer and a 2-neuron output layer. Each hidden neuron receives 8 inputs, from: 5 distance sensors, 1 color signal and 2 motor feedbacks, and uses the following standard Gaussian function as activation function:

$$y = e^{-\frac{(\|\boldsymbol{x}-\boldsymbol{c_i}\|)^2}{b_i^2}}$$

All input-hidden-layer weights are set to 1, but within its activation function each hidden neuron conveys 9 variable parameters of its own. Of these, 8 parameters, confined all of them to the range [0, 1], constitute the center vector $\boldsymbol{c}_i$ of neuron i; while the remaining parameter b_i corresponds to the width, also known as spread or amplitude, of the Gaussian function and its value is bounded between 0.05 and 2.5. Meanwhile, each of the 2 neurons of the output layer receive the 5 neuron outputs from the hidden layer plus 1 bias input; the weights of this 12 synaptic connections are restricted to the interval [-5, 5]. In addition, these units have a linear activation function and their outputs are linearly scaled between 1300μs and 1700μs before they are entered to the motor lookup table.

2.3.4. Fully Connected Recurrent Neural Network

Within this network, each neuron is connected to every input and to every other neuron. This network has 6 sensory inputs (5 distance sensors and 1 color signal) and 7 sigmoid units, 2 of which

are the motor outputs. As it is required by the simulation, the two outputs are eventually scaled from [0, 1] to [1300μs, 1700μs]. All 91 synaptic weights are bounded between -5 and 5.

2.3.5. Continuous-Time Recurrent Neural Network

The topology of this network is exactly the same as the topology of the Fully Connected Recurrent Neural Network. The only difference between the two concerns the activation function used by the neurons of each network. Instead of a sigmoid function, every neuron of this network has the following dynamic activation function:

$$\frac{dy_i}{dt} = \frac{1}{\tau_i}\left(- y_i + \sum_{j=1}^{N} w_{ji} \cdot \sigma(y_j + \beta_j) + \sum_{k=1}^{S} w_{ik} \cdot I_k \right)$$

This dynamic function is computed using Forward Euler method with a step of 0.1 seconds. Each neuron is associated with a time constant τ, a bias β, and 13 postsynaptic weights, whose values are confined to the intervals [1s, 50s], [-1, 1] and [-5, 5] respectively. Furthermore, since the dynamic activation function does not constrained its output to an upper or lower bound, the outputs of the two motor neurons are first sent to a sigmoid function in order to limit their values between 0 and 1.

2.3.6. Echo State Network without leak rate

This network is composed of a reservoir consisting of 100 neurons and an output layer consisting of 2 neurons. All units are governed by a hyperbolic tangent function. Both the reservoir and the output layer receive inputs from 5 distance sensors, 1 color signal, 2 servomotors and 1 bias signal. Once again, the two output signals are linearly scaled between 1300μs and 1700μs before they are entered to the motor lookup table. The reservoir weight matrix is randomly initialized with a spectral radius of 0.9, a connectivity of 0.1, and fixed weights of value 1. Likewise, the input-reservoir and input-output weight matrices are also randomly initialized with a connectivity of 0.3 and fixed weights of value 0.25. Lastly, the reservoir is fully connected to the output layer by means of 200 synapses, whose weights are confined to the range [-5, 5]. This network has the following state update equations:

$$x(t+1) = \tanh(\boldsymbol{W}_{res}^{\boldsymbol{res}} \cdot x(t) + \boldsymbol{W}_{in}^{res} \cdot u(t))$$

$$y(t+1) = \tanh(\boldsymbol{W}_{\boldsymbol{res}}^{\boldsymbol{out}} \cdot x(t+1) + \boldsymbol{W}_{\boldsymbol{in}}^{\boldsymbol{out}} \cdot u(t))$$

Where ***x*** denotes the state of the reservoir, *u* is the input vector, ***y*** represents the state of the output layer, and $\boldsymbol{W}_{res}^{res}$, $\boldsymbol{W}_{in}^{res}$, $\boldsymbol{W}_{res}^{out}$, $\boldsymbol{W}_{in}^{out}$, are weight matrices [9].

2.3.7. Echo State Network with leak rate

This network is virtually identical to the ESN network without leak rate. The two differ only in their reservoir state update equations. The ESN+α network employs a modified version of the equation used by the ESN network, redefined by adding to it a leak rate term *α*, ranging from 0 to 1. The resulting equation is:

$$x(t+1) = \tanh((1-\alpha) \cdot x(t) + \alpha \cdot (\boldsymbol{W}_{\boldsymbol{res}}^{\boldsymbol{res}} \cdot x(t) + \boldsymbol{W}_{\boldsymbol{in}}^{\boldsymbol{res}} \cdot u(t)))$$

2.3.8. Spiking Neural Network

This network comprises a total of 10 neurons which are fully connected to each other. The input layer consists of 6 sensory inputs (5 distance sensors and 1 color signal) and 1 bias. Once every 100ms the current values of the 7 inputs are transformed into spikes with a firing rate proportional to their magnitudes; being these spike trains the ones that are ultimately read by the neural processing units. On the other hand, the output layer contains 4 neurons, whose outputs are used to govern the two servomotors; two of these neuron outputs set the positive speed signals relative to each actuator and the other two set the corresponding negative speed signals. Similarly to the case of the sensory inputs, these speed signals are updated only once every 100ms, and they are defined, in absolute value, as the number of spikes fired by the corresponding neuron in the last 20ms of the 100ms time-window. Ultimately, to simulate the PWM signals that would effectively drive the servomotors, the positive and negative speed signals associated with each actuator are added together; and afterwards, the aggregated speeds are linearly scaled to the range [1300μs, 1700μs]. Meanwhile, unlike the incoming and outgoing signals, the state of this neural network is updated every 1ms. The state of each neuron is determined by a threshold *Θ* and a membrane potential v_i; whenever the membrane potential

exceeds the threshold, neuron *i* fires a single spike at that time step. The threshold is a constant equal to 0.1; whereas, according to the applied Spike Response Model [10],[11] the membrane potential is defined as follows:

$$v_i(t) = \sum_j w_{ij} \sum_f \varepsilon_j(s_j) + \sum_f \eta_i(s_i)$$

$$\varepsilon(s) = e^{-[(s-\Delta)/\tau_m]} \cdot (1 - e^{-[(s-\Delta)/\tau_s]})$$

$$\eta(s) = -r \cdot e^{-(s/\tau_m)}$$

Where, w_{ij} are the synaptic weights of the network, s_n denotes the difference between the current time and the time in which neuron or input *n* last fired a spike, Δ is a constant of value 2ms that symbolizes the underlying spike transmission delay, τ_s and τ_m are the synaptic and membrane time constants which have values of 10ms and 4ms respectively, *r* is a uniformly random value between 0 and 1, and *f* denotes each presynaptic spike relative to neuron *i*. Furthermore, if the argument *s* of functions $\varepsilon(s)$ and $\eta(s)$ lies outside the intervals $\Delta \leq s \leq 20ms$ and $0 \leq s \leq 20ms$ respectively, the result of the corresponding function is set to zero. With regard to the synaptic connections, they all have three properties: weight, sign and activation state. The weight of every synapse is set to the same constant value of 1. The sign of each synapse can be either positive (+1) or negative (-1); this depends respectively on the excitatory or inhibitory potential of the neuron that engenders the postsynaptic connection. In turn, within the simulation this postsynaptic potential is considered a parameter specific to each neuron, that can be granted values of -1 (inhibitory) or +1 (excitatory). All inputs are thought of as having an excitatory potential. Finally, concerning the activation state, this is regarded as a property or parameter specific to each synapse; a value of 1 indicates that the connection exists, while a value of 0 corresponds to an inactive connection.

2.3.9. Spike-Timing Dependent Plasticity

Spike-Timing Dependent Plasticity (STDP) is a mechanism which potentiates or depresses a synapse, i.e. increases or decreases the strength of its synaptic weight, by means of a plasticity rule that depends on the relative timing between single presynaptic and postsynaptic spikes [12],[13]. STDP was implemented over the previously described Spiking Neural Network and Spike Response Model, but with the adjustment that the synaptic weights are no longer fixed to 1; instead each one of them is randomly set at beginning of every epoch within a range of 0 to 1. Moreover, these weights do not remain constant during the simulation of the neurocontroller; to the contrary, they are allowed to change at each iteration according to a plasticity rule. Said plasticity rule operates at the synapse level; thus, it is computed independently for every neural connection and depends on 5 parameters γ, τ^+, τ^-, A^+ and A^-, which are unique to each synapse and that will be explained next. Every synaptic weight is updated according to the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta_d w_{ij}(t)$$

Where $\Delta_d w_{ij}$ indicates the absolute weight change between the current ant the next iteration, which is determined thanks to the following directional damping approach:

$$\Delta_d w_{ij}(t) = (1 - w_{ij}(t)) \cdot \Delta w_{ij}(t), \forall \Delta w_{ij} \geq 0$$

$$\Delta_d w_{ij}(t) = w_{ij}(t) \cdot \Delta w_{ij}(t), \forall \Delta w_{ij} < 0 .$$

Furthermore, $\Delta w_{ij}(t) = \gamma \cdot \zeta_{ij}(t)$, where γ is a learning rate constrained to the interval [0.001, 0.5], and ζ_{ij} refers to the STDP plasticity rule defined as follows:

$$\zeta_{ij}(t) = P_{ij}^+(t) \cdot f_i(t) + P_{ij}^-(t) \cdot f_j(t)$$

Where P_{ij}^+ and P_{ij}^- are functions that record the postsynaptic spikes f_i or the presynaptic spikes f_j, respectively, emitted by the neurons connected by synapse *ij*. In turn, these two functions are governed by the following equations:

$$P_{ij}^+(t) = P_{ij}^+(t-1) \cdot e^{-\frac{\delta t}{\tau^+}} + A^+ \cdot f_j(t)$$

$$P_{ij}^-(t) = P_{ij}^-(t-1) \cdot e^{-\frac{\delta t}{\tau^-}} - A^- \cdot f_i(t)$$

Where τ^+ and τ^- denote exponential decay constants confined to [1ms, 40ms], A^+ and A^- are dimensionless parameters, ranging from 0.001 to 2, and δt corresponds to the simulation time step, which is 1ms.

2.4. Evolutionary Algorithm

In this work, a genetic algorithm was used in order to evolve the desired solutions for the proposed experiment. Under this approach, the variable parameters that define the behavior of the robotic agent were organized into a single vector of values, also known as gene sequence or chromosome, where each parameter equates to one gene. Different sets of values create different chromosomes, and these chromosomes are the structures that are taken as inputs by the genetic algorithm.

2.4.1. Encoding

Each chromosome encodes only the controller of a robotic agent, i.e. each chromosome is a vector containing a set of values corresponding to the variable parameters of the neural network set to act as the controller of the agent. Within a chromosome, these parameters are arranged in no particular order, although, generally, parameters belonging to the same category (synaptic weights, learning rates, time constants, etc.) were placed together.

It is worth noting that as described in 2.3 the topologies of all neural architectures being used were a priori fixed. Furthermore, they were not allowed to evolve or change in any way during the evolutionary process; thus, all chromosomes have a static length.

An 8-bit binary encoding is used for the construction of the chromosomes. Nevertheless, most of the variable parameters to be encoded are expressed as real numbers. This discrepancy was reconciled by first scaling the real numbers between [0, 255] or [-127, 127], while taking into account the specific range of values of each parameter. Next, the scaled values were rounded to the nearest integer. And finally, the resulting integers were converted from a decimal to a signed or unsigned binary notation, correspondingly. In the case of the Spiking Neural Network, some of its variable parameters are inherently binary; as such, they are encoded each in one bit of the chromosome.

The chromosomes encoding SLP, MLP and FRNN controllers are composed of only real-value synaptic weights; thus, they comprise 18, 57 and 91 genes respectively. The chromosomes that represent RBF controllers are built from 5 center vectors, comprising 8 parameters each, 5 widths and 12 synaptic weights; making a total of 57 real-value genes. The chromosomes associated with a CTRNN architecture, composed of 7 neurons, comprise 7 time constants, 7 biases and 91 synaptic weights, all of them real-value parameters. The chromosomes encoding ESN or ESN+α controllers encompass 200 reservoir-output weights and a learning rate, if appropriate; all other parameters are fixed at the beginning of each evolutionary process. The chromosomes representing SNN agents are made up of 170 synaptic activation states and 10 postsynaptic potentials, totaling 180 binary genes. Lastly, the chromosomes that encode SNN+STDP controllers take the same 180 binary genes comprised in the chromosomes associated with a SNN architecture plus 850 real-value genes, which correspond to the 170 synapses embedded in the SNN+STDP architecture and the 5 variable parameters that define each synapse, namely γ, τ^+, τ^-, A^+ and A^-.

2.4.2. Genetic Algorithm

The evolutionary optimization loop uses a well-known Multi-Objective Genetic Algorithm named NSGA-II (Non-dominated Sorting Genetic Algorithm-II) [14], which is executed for a maximum of 300 generations. It has a constant-size population of 100 individuals. Half of the population is uniformly randomly initialized at the start of each run by assigning a value 0 or 1 with probability 0.5 to every bit of each chromosome; the other half is simply the bitwise negation of the first half. Each chromosome encodes a neural network by means of an 8-bit binary representation, except for the SNN and SNN+STDP networks that possess binary parameters; thus no encoding is required in those cases. This genetic algorithm assigns a multi-objective fitness to each individual of the parent and offspring populations by testing and rating its performance in the T-maze alternation task; and afterwards, based on these performances, the algorithm employs a Pareto dominance ranking scheme in order to enforce a fitness-based evolutionary pressure during the selection and reinsertion steps. Furthermore, the genetic algorithm uses binary tournament selection for defining a mating pool of 20 individuals, unrestricted mating, uniform crossover with

mixing ratio of 0.2, non-adaptive binary mutation by flipping with mutation rate of 0.02 per bit, and elitist fitness-based reinsertion.

3. Experimental Study

3.1. Experimental Setting

3.1.1. The Task

The experimental task is the T-maze alternation task. A robot is given 16 epochs and 2 trials each epoch to traverse the maze, whose goal is switched from arm to arm between epochs, but it remains fixed between trials. For each epoch, the task is solved when the robot reaches the goal without colliding with the walls of the maze while satisfying the condition during the second trial that the robot must move directly towards the goal, i.e. upon reaching the junction the robot must choose correctly the arm of the maze where the goal is located.

The reasoning behind choosing the T-maze alternation task in the present work rests on the notion that it is far from trivial to know a priori if a given task can be solved or not by a purely reactive agent; in other words, an agent lacking memory skills. Moreover, this difficulty is exacerbated due to the fact that evolutionary algorithms are especially proficient in finding these trivial reactive solutions. Nonetheless, empirical evidence reported in [4] and [5] suggests that tasks performed in dynamic environments favor agents that possess a working memory. Since in the T-maze alternation task the location of the goal of the maze does not remain static, instead it is changed between epochs, from left arm to right arm or vice versa, then said task is regarded to take place in a dynamic environment; and hence, it is expected that it will encourage the development of working memory behaviors.

3.1.2. The Controllers

As explained in the introduction, the challenges posed by the evolution of a working memory were taken as an opportunity to research the competence of 9 neural models to engender controllers that manage to solve the T-maze alternation task. The topologies of these architectures were explained in detail in 2.3.

These models are Recurrent Single Layer Perceptron (SLP), Recurrent Multilayer Perceptron (MLP), Recurrent Radial Basis Function Network (RBF) Fully Connected Recurrent Neural Network (FRNN), Continuous-Time Recurrent Neural Network (CTRNN), Echo State Network without leak rate (ESN), Echo State Network with leak rate (ESN+α), Spiking Neural Network (SNN) and Spiking Neural Network with Spike-Timing Dependent Plasticity (SNN+STDP).

The above architectures were chosen due to their widespread use in the field of Evolutionary Robotics, and because together they comprise three generations of increased realism in the simulation of neural networks.

3.1.3. The Genetic Algorithm

The variable parameters of each of the 9 neural models being studied need to be tuned first in order to attempt to produce agents that display a working memory. This is accomplished by using the genetic algorithm specified in 2.4.2, which makes use of a multi-objective fitness function designed specifically for rating the performance of every individual within the parent and offspring populations of every generation at solving the T-maze alternation task.

Expanding on the description of said function, it comprises 5 objectives: 1) average distance traveled between the 32 trials, 2) maximum distance traveled over the 32 trials, 3) average number of collisions between the 32 trials, 4) minimum number of collisions over the 32 trials, and 5) number of times the agent moves directly towards the goal of the maze in the second trial of each of the 16 epochs.

Each of the first two objectives can be considered on their own as tailored fitness functions, because a computational routine was especially hand-formulated in order to calculate the distance traversed by the robot through the maze, while disregarding backwards or sideways walks. As a result, the maximum distance that an agent could travel in one trial is 40 cm, but if it reaches the goal it is granted with a fitness of 100 cm. Individually, the next three objectives can be regarded as aggregate fitness functions, since their purpose is to count the number of high-level successes or failures.

3.1.4. The Simulation

The task, including environment, robot and controller, is simulated entirely in a virtual world. Each trial has a maximum simulation time of 10 seconds, but it could end earlier if the agent reaches the goal. The simulation time step is 0.1 seconds at all cases, except for the neurocontrollers associated with the SNN and SNN+STDP architectures, which were simulated with a time step of 1ms. The simulation is restarted between epochs, but it is kept running between trials. In this context, the term restart refers to resetting back to zero the speeds of the servomotors as well as all internal states of the neural networks. With regard to the evolutionary algorithm, it is run 20 times per architecture and for 300 generations each time.

It is worth noting that limiting the scope of this work to a virtual environment made it possible to make some simplifications to the physical simulation, as stated in 2. Nevertheless, it is the view of the author that these simplifications do not undermine the results achieved, since the purpose of this study is to assess the feasibility of the neuroevolutionary approach for developing working memory behaviors, and the emergence of said behaviors will or will not take place regardless of whether the agent is real or virtual.

3.2. Experimental Results

Table 3 summarizes the results obtained for each neural model. The following three metrics were used to evaluate each architecture:

- Final population performance (FPP): This metric refers to the mean performance of all individuals within the final populations of the 20 evolutionary runs executed. In turn, the performance of each individual is defined as the number of times out of 16 epochs that the agent moves directly towards the goal of the T-maze in the second trial of an epoch.
- Best agent performance (BAP): This metric indicates the number of times out of an extra 100 epochs, performed during a post-evolutionary analysis, that the best agent evolved over the 20 evolutionary runs executed moves directly towards the goal of the T-maze in the second trial of an epoch.
- Network performance (NP): This metric concerns with the number of evolutionary runs out of 20 in which working memory behaviors have emerged.

Neural Models	FPP	BAP	NP
SLP	-	50	0
MLP	-	50	0
RBF	-	50	0
FRNN	15.69	100	15
CTRNN	15.87	100	20
ESN	-	50	0
ESN+α	15.81	100	18
SNN	-	50	0
SNN+STDP	7.73	87	20

Table 3. Performances of the examined models.

The above table evidences that out of the nine neural architectures that were tested only four of them, namely the FRNN, CTRNN, ESN+α and SNN+STDP models, were capable of evolving at least once working memory behaviors. In turn, out of these four neural models only the CTRNN and SNN+STDP architectures proved to be 100% effective for fostering the evolution of a working memory. It can also be noticed that the SNN+STDP model showed a BAP performance of merely 87%, unlike the perfect score reported by other neural models. Conversely, the FRNN and ESN+α architectures favored the evolution of working memory behaviors in 75% and 90% of the 20 evolutionary runs executed, respectively.

4. Discussion

4.1. Underlying Mechanisms of Neuroevolution

Figure 9 depicts, for each of the four neural architectures that succeeded in developing working memory behaviors, the gradual progress of its population performance metric through the generations of one typical evolutionary run.

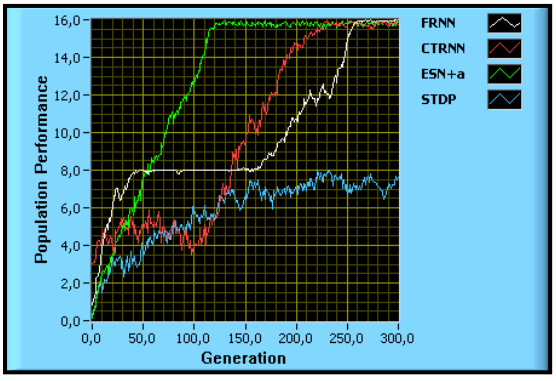

Figure 9. Population performance progress.

The population performance metric measures the mean performance of all individuals within a given generation; in turn, the performance of each individual is defined as the number of times out of 16 epochs that the agent moves directly towards the goal of the T-maze in the second trial of an epoch.

The first observation that can be drawn from Figure 9 is that different neural models allow for different evolutionary rates of improvement, and that these rates depend strongly on the size of the chromosomes that encode the neural networks. This phenomenon is particular evident during the first 30 to 50 generations, a period in which reactive agents overrun the populations. During these first generations the FRNN model which has the smallest chromosome with 91 real-value genes exhibits the fastest convergence rate, said architecture is followed by the CTRNN (105 real-value genes), the ESN+α (200 real-value genes) and the SNN+STDP (850 real-value and 180 binary genes) models, in that order.

Furthermore, Figure 9 provides plenty of information on how the evolution of a working memory took place over the different architectures and which hurdles they encountered along this evolutionary process. With regard to the FRNN architecture, initially it favored the evolution of reactive behaviors, i.e. navigation skills, which immediately took over the entire population. After the reactive agents had reached their maximum memory-related fitness of 8, the population performance remained stagnant for many generations; over 100 generations in the recorded run. From the 1st up to the 150th evolutionary step the FRNN architecture did not reveal any signs of evolving working memory behaviors, but immediately after the 150th generation, most likely due to a fortunate mutation, individuals showing incipient memory skills started to emerge within the population. Because of them, the evolutionary progress was set back in motion, leading eventually to a rapid-paced evolution of memory-endowed agents. Based on the evidence at hand, it can be inferred that the FRNN model suffered from a local optima problem, in which the lack of diversity mechanisms associated with the evolutionary algorithm allowed the rapid propagation throughout the population of relatively fitter purely reactive behaviors, with the detrimental result that a thorough exploration of the search space was mostly halted after that. This issue could be tackled by adding into the genetic algorithm niching, crowding, adaptive mutation or novelty search strategies, applied to either the genotypic or phenotypic space or to both [15],[16].

On the other hand, in reference to the CTRNN model, since the beginning of the evolutionary run half of the population was comprised of reactive agents showing outstanding navigational skills, but completely lacking memory skills, and the other half consisted of non-reactive solutions showing remarkable memory skills, but incipient navigational skills. Despite this propitious start, the population performance of this network also remained stagnant and even decreased for many generations. During that period, an intense competition between the previously described two groups of individuals with different sets of skills took placed, neither the memory-endowed solutions were able to optimize their navigational skills, nor the reactive solutions succeeded in evolving a working memory. This situation continued until approximately the 100th generation when promising hybrids were bred; and as a result, the convergence of the algorithm towards the development of more robust working memory behaviors was resumed.

The SNN+STDP architecture underwent a similar evolutionary process as the CTRNN model, but since its chromosome is roughly eight times larger than the chromosome of the CTRNN architecture, the genetic algorithm needs proportionally more generations to evolve robust working memory behaviors; that being the reason the SNN+STDP architecture showed only a 87% BAP performance. In addition, as both, the CTRNN and the SNN+STDP models, have experienced significant stagnation in their evolutionary processes, these architecture would also benefit greatly from the previously proposed niching, crowding or novelty-search diversity maintaining mechanisms.

Finally, the ESN+α model displayed an ideal evolutionary progress. Its population performance converged linearly from start to end. This came to being because the evolving solutions were able to easily incorporate navigational and memory skills to their overall

behaviors. But more intriguing than this is the fact that this architecture did not achieve a perfect network performance despite having no difficulties in developing working memory behaviors in most evolutionary runs. Given the generally steady progress of the population performance of this neural model, it is unlikely that the abnormalities observed would arise from underlying problems related to the search space or to the evolutionary algorithm. Therefore, it can be hypothesized that these abnormalities are linked to the input-reservoir and reservoir-reservoir weight matrices, which are fixed at the beginning of each evolutionary run and are shared by every individual throughout the generations. If, in some instances, these a priori fixed matrices lacked the inherent properties necessary for engendering a working memory, that would certainly account for the absence of memory-endowed agents in the final populations of 2 out of the 20 evolutionary runs that were executed.

4.2. Temporal or Spatial Working Memory

One approach for solving the T-maze alternation task is to use only temporal cues; for instance, if an agent has the ability to sense time, it could decide, as it reaches the junction of the maze, whether to turn right or left depending on its knowledge of the time that has passed since the start of the current epoch until that instant. In practice, this strategy would obey the following logic; assuming, for example purposes, that for short elapsed times since the beginning of an epoch the default behavior of the robot performing the task is to turn right at the junction of the T-maze and for longer elapsed times is to turn left; then, in the first trial of a given epoch the robot will of course turn right at the junction of the maze. If the goal is located precisely in the right arm of the maze, the robot will arrive there quickly, but if the goal is located in the opposite arm, the robot will either wander around until the simulation time of the first trial ends or it will traverse the maze from one arm to the other until eventually arriving at the goal. Immediately after the robot reaches the goal or the simulation time is over, the robot is replaced to the starting position; an event that marks the beginning of the second trial of the epoch. From there the robot will again move towards the junction of the maze; once at the junction it should use its sensing of time in order to move directly towards the goal of the maze by turning right if the duration of its journey so far is regarded as short, i.e. it falls below a threshold value, which occurs when the goal is located in the right arm, and consequently, the robot reaches it rapidly in the first trial; or by turning left if the duration of its journey until that moment is above that same threshold value, which happens when the goal is located in the left arm, and thus, the robot must further explore the T-maze during the first trial.

Another approach for solving the proposed task is to use only spatial cues; for example, an agent could base its decision of turning right or left at the junction of the maze on its memory of the floor color sensed at the end of the arm it last visited. One way to implement this approach is for a robot moving through the T-maze to turn right at the junction if it does not have the memory of previously visiting the wrong arm, i.e. the arm where the goal is not located; and conversely, to turn left if it recalls visiting the wrong arm. In turn, in order to develop a memory of visiting the wrong arm of the maze, the robot could remember that as it reached the furthest wall of the visited arm its front distance sensor registered a high value while simultaneously its color sensors detected a white surface. It is worth noting that to effectively associate these sensor readings to a visit to the wrong arm of the maze, the exploration strategy of the robot should avoid visiting during its journey any other instances that would generate the same readings. In practice, in the first trial of a given epoch, when a robot following the previously described control rule reaches the junction of the maze, it will turn right because it has no prior memory of having visited any arm of the maze. If the goal is located precisely in the right arm, the first trial of the epoch will end right then, without the robot having acquired the memory of visiting the wrong arm of the maze; thus, in the next trial, the automaton will steer straight towards the goal by turning right for a second time at the junction. However, if the goal is located in the left arm, by initially turning right at the junction the robot will reach the wrong arm of the maze and it will create a memory of this event; consequently, in the second trial, the automaton will likewise move directly towards the goal by choosing this time to turn left at the junction, i.e. the opposite direction to where it turned in the first trial.

The proposed T-maze alternation task could also be solved by applying a spatiotemporal approach; for instance, an agent capable of remembering how long the first trial of an epoch lasted could use this temporal information to choose whether to turn right or left at the junction of the maze in the second trial of the same epoch. This is regarded as a spatiotemporal approach, since on the one hand the agent must rely on spatial cues to detect the end of the first trial, and on the other hand it must sense the passage of time from the beginning to the end of the first trial. The strategy employed by an agent following a spatiotemporal approach would be similar to the one described previously regarding an agent following a purely temporal approach; for example, it could turn right at the junction of the T-maze if the duration of the preceding trial was short and left if the duration was long. According to this strategy, in the first trial of an epoch a robot will steer right at the junction of the T-maze, because its memory of how long the previous trial lasted is initialized to zero at the beginning of every epoch. If the goal is located in the right arm of the maze, the robot will reach it quickly, but if the goal is located in the left arm, the robot will take more time than in the previous scenario to arrive at the goal; by the end of the trial the robot will have stored either the short or the long duration of it in its memory. Afterwards, in the second trial of the epoch, the robot will use the temporal information recorded in its memory in order to turn right once again at junction of the maze if the duration of the first trial was short, or to turn left otherwise.

In order to assess which approach, whether a temporal, spatial or spatiotemporal one, was used by the evolved controllers, the agents that showed the best performances out of each neural architecture that successfully foster the evolution of a working memory were tested in modified environments and for varying simulation times. Figure 10 portrays the behaviors displayed by a CTRNN controller under different simulation times for the first trial of an epoch (experiments with FRNN, ESN+α, and SNN+STDP controllers all produced similar results).

It can be noticed from Figure 10 that if an agent is given less than 110 simulation steps during the first trial of an epoch, it will always steer right at the junction of the maze in the second trial; whereas if the simulation time exceeds 130 steps during the initial trial, the robot will invariably move straight towards the left arm of the maze in the final trial. This analysis indicates that the behavior of the tested agents was highly conditioned by the simulation time instead of by the unusual sensor and motor feedback readings to which they were exposed.

In conclusion, the results of this supplementary experiment reveal that the memory skills displayed by the best evolved agents are consistent with a temporal rather than with a spatial or spatiotemporal working memory and that, at least for the case of these agents, a temporal working memory behavior rely on sensing the passing of time.

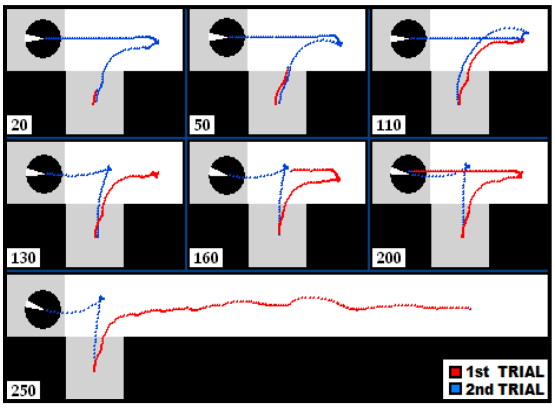

Figure 10. Response of an evolved CTRNN agent to varying environments and simulation times.

4.3. How do agents sense time?

In order to investigate the underlying mechanisms that allow evolved individuals to sense time, the sensory and neural signals pertaining to a memory-endowed agent (CTRNN architecture) were recorded while it was performing the T-maze alternation task. These recordings comprised a normal behavior (Figure 11 and Figure 12) as well as an alternate behavior (Figure 13 and Figure 14).

An agent is said to execute a normal behavior when from the starting position it directs itself towards the same arm of the maze in both trials of an epoch. Conversely, an agent is said to execute an alternate behavior when it moves straight to one arm of the maze in the first trial and straight to other arm in the second trial. Figure 15 exemplifies both behaviors.

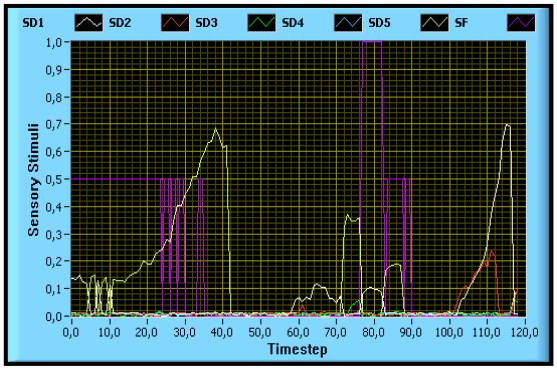

Figure 11. Sensory stimuli (normal behavior).

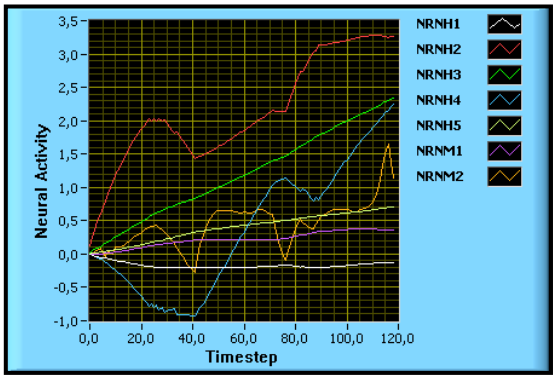

Figure 12. Neural activity (normal behavior).

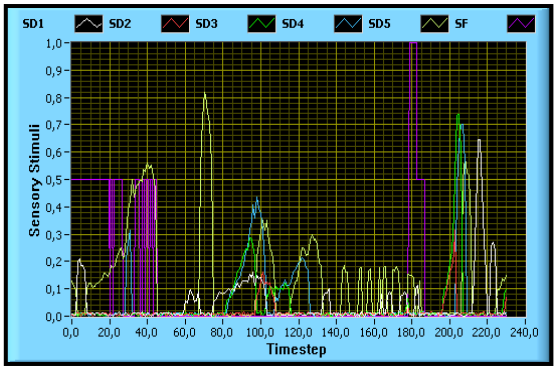

Figure 13. Sensory stimuli (alternate behavior).

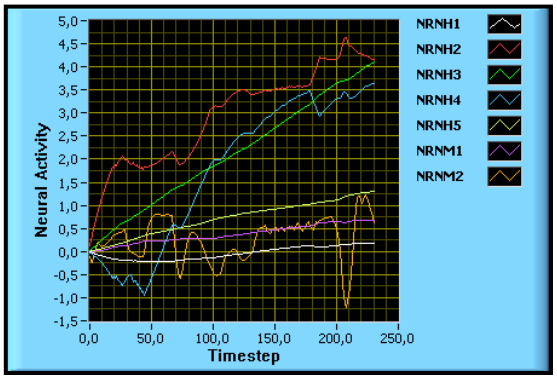

Figure 14. Neural activity (alternate behavior).

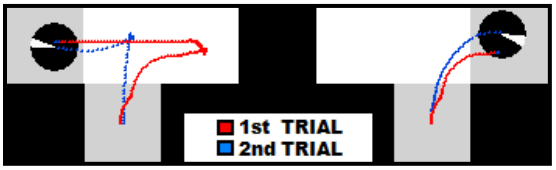

Figure 15. Alternate (left) and normal (right) behaviors.

The previous recordings reveal that the overall behavior of the examined agent can be summarized by means of three neural mechanisms. First, in the absence of sensory stimulation the state of motor neuron M2, connected to the left servomotor, converges rapidly from any previous value to a positive resting potential of approximately 0.7. Second, M2 is highly sensitive to sensory stimulation, especially to the lateral distance sensor on the right side of the robot; in the presence of stimuli the state of this neuron converges abruptly to a negative value proportional to the intensity of the corresponding sensory stimulation. Third, the state of motor neuron M1, connected to the right servomotor, is unaffected by sensory stimuli; it grows almost linearly throughout the simulation from an initial value of zero.

Based on these three mechanisms, the behavior of a memory-endowed agent is described as follows. The output of M1 and the resting state of M2 can be regarded as control and threshold signals respectively. When a robot begins the first trial of an epoch, the control signal has values close to zero, while the threshold signal converges rapidly to positive values; this difference causes the robot to turn towards the right arm of the maze. If the goal is located in that arm, the robot will be immediately repositioned to the start position; as a result, the difference between the threshold and control signals will remain virtually unchanged, and consequently, in the second trial the robot will once again steer in the direction of the right arm of the maze. This explains the normal behavior. However, if during this first trial the goal is located in the arm opposite to where the robot directed itself first, the automaton will have to turn around and traverse the maze until it reaches the goal. This extra time needed by the robot to further explore the maze during the first trial of an epoch allows the control signal to grow up to the level of the threshold signal. As a consequence of this, at the beginning of the

second trial the agent will have the tendency to move straight forward until its front distance sensors SD3 and SD4 detect the wall at the junction. At that moment, due to its high sensitivity to sensory inputs, the state of motor neuron M2, i.e. the threshold signal, drops instantly to a negative value; and on the other hand, the control signal remains unchanged. This newly generated difference between the control and threshold signals induces the robot to rotate counterclockwise towards the left arm of the maze. Finally, as soon as the sensory stimulation is stopped, neuron M2 will immediately return to its resting potential; correspondingly, the threshold signal will return to its previous positive value close to the value of the control signal, causing the robot to move straight ahead towards the goal.

From the previous description it is clear that in order to sense time the evolved agents rely on a time-synchronized neural signal, i.e. the previously dubbed control signal. For this particular experimental setting, said signal is required to be in one state at the beginning of the task and in a different state towards the end of the task, while remaining at all times insensitive to sensory disturbances. In addition, since sensory stimuli do not play a role in the generation of the aforementioned time-synchronized signals, it ultimately follows that these originate solely from the internal dynamics of the neural architecture being employed. It is worth mentioning that the evolved agents (CTRNN architecture) able to solve the T-maze alternation task made use of a ramp-like control signal, but several other signals, such as a delayed step signal, could have served the same purpose. Nevertheless, as agents are faced with more difficult memory tasks, for instance a multiple T-maze task, a linear output signal is expected to be the preferred method to sense time.

This newly acquired understanding that memory skills arise from time-synchronized signals, which in turn, originate from the internal dynamics of the neural model being use, led to the hypothesis that the network performance of the tested neural architectures is linked to their potentiality to produce such signals, especially linear output signals. In order to test this hypothesis, a further experiment was conducted in which the variable parameters pertaining to a single self-recurrent neuron were randomly set 10.000 times, each time the neuron state was simulated during 10 seconds, after which the approximation of its output signal to a linear function was tested using the Mean Squared Error (MSE) measure. It is worth noting that recurrency is imperative for attaining signals synchronized with time. Five different activation functions were tested, including the sigmoid function (specifically the logistic function), the hyperbolic tangent function, the hyperbolic tangent function with leak rate, the dynamic function of the CTRNN architecture and the Spike Response Model (SRM) of the SNN model. All of them, except the dynamic function, were tested with a connection to a bias input of value 1. The radial basis function was excluded from this experiment because the topology conceived for the RBF architecture makes it impossible for a single neuron to detach itself from sensory stimuli. The results obtained validate the initial hypothesis. The dynamic function generates a ramp-like output signal for nearly any random set of variable parameters, although time constants over 5 yield the best results; correspondingly, the CTRNN architecture had a perfect network performance. The hyperbolic tangent function with leak rate is also highly likely, though not at the same level of the dynamic function, to generate an almost linear output signal from a given random set of variable parameters, especially for leak rates under 0.2, and self-recurrent synaptic weights in the range ±0.9; this finding agrees with the outstanding performance showed by the ESN+α model. Nevertheless, the fact that the synaptic weights and topology needed to produce a nearly linear output signal were randomly fixed at the beginning of each evolutionary run, is the most plausible reason behind the flawed performance of the ESN+α architecture, since it is highly probable that this initialization sporadically engendered reservoirs lacking nearly linear signals.

On the other hand, the hyperbolic tangent and the logistic functions originate an almost linear output signal only for very specific configurations of the self-recurrent and bias synaptic weights. With regard to the hyperbolic tangent function, bias weights in the range of ±0.2 and self-recurrent weights between 0.98 and 1.02 produce excellent results. In addition, as it was the case with the ESN+α architecture, these

input-reservoir and reservoir-reservoir synaptic weights were not allowed to be tuned by the evolutionary algorithm. As a consequence of these two adverse conditions, it is highly unlikely that the reservoirs initialized prior to the start of each evolutionary run produced nearly linear signals; accordingly, the ESN architecture never promoted the development of agents endowed with memory skills. One way to improve the performance of this neural architecture would be to implement a two-stage incremental evolutionary approach. Under said approach, in the first evolutionary stage the algorithm would optimize only the input-reservoir and reservoir-reservoir weight matrices with the goal of attaining almost linear output signals from the reservoir neurons. During the second evolutionary stage, only the reservoir-output synaptic weights would be optimized, as it is the case with the evolutionary approach currently used, but this time the weight matrices evolved during the first evolutionary stage would replace the randomly initialized matrices employed in the current approach.

Concerning the logistic function, bias weights between -2.05 and -1.95 along with self-recurrent weights in the interval -4.05 to -3.95 generate a quasi-linear output signal; additionally, bias weights in the range of -2.25 and -2.05 originate step-like output signals for windows of self-recurrent weights of approximately 0.1 centered in values lying between -5 and -4.05 (the value of -5 corresponds to the self-imposed upper bound of the synaptic weights). Despite these limitations, the FRNN architecture showed a remarkable network performance, which demonstrates the extreme expediency of evolutionary algorithms for finding non-trivial solutions. Notwithstanding, similarly to the ESN model, a good measure to enhance the performance of the FRNN architecture would be to incorporate an extra objective to the fitness function of the evolutionary algorithm, which rewards the development of nearly linear output signals. The SLP and MLP architectures used the same logistic activation function as the FRNN model, but surprisingly the network performances of the former two were far worse than the performance of the latter. It is suspected that the reason for this is that unlike the FRNN architecture every recurrent loop present in the SLP and MLP topologies necessarily goes through the output layer; thus, every time-synchronized signal that might have emerged was directly used to drive the servomotors, without being previously scaled to more manageable speeds. In keeping with this reasoning, the performances of the SLP and MLP architectures could be enhanced by altering their topologies with the aim of making them more akin to a FRNN network.

Finally, under this particular experimental setting a SRM neuron produces a sequence of serially uncorrelated random values (white noise). Due to their complexity, the neural mechanisms underlying the performance of the SNN and SNN+STDP architectures were not further studied during the present work. Thus, their understanding remains a pending task to be tackled by the author in future endeavors.

5. Conclusion and Future Work

This work has validated the feasibility of generating working memory behaviors in robotic agents by means of a neuroevolutionary approach. It is worth mentioning that the evolved agents showed an outstanding robustness for solving the T-maze alternation task. Out of 100 epochs the best agents were able to solve properly the task 100% of the times (Table 3), even under highly noisy environments. However, it is left for further studies to investigate if this robustness can be extended from virtual to real agents.

Nevertheless, this evolutionary approach was not free from difficulties. A recurrent problem was that several times a few mediocre solutions spread rapidly and rampant throughout the entire population, causing artificial evolution to converge prematurely to local optima and to remain there for many generations. In future work, it is hoped to research different methods such as niching, crowding or adaptive mutation that would maintain the diversity of the population or that would increase said diversity as needed.

Furthermore, the local optima problem arises not only from the lack of population diversity, but also from the lack of a proper selective pressure. In this work, a multi-objective fitness function accounted for this pressure; it comprised as one of its objectives the number of times out of 16 epochs an agent remembered the location of the

goal it had previously visited. As such, this aggregate objective requires an agent to solve fully a complex task in order to increase its fitness, i.e. there are no intermediate rewards. This situation eventually led many times to a bootstrap problem, since randomly initialized genotypes or random mutations of a purely reactive agent were not able to solve such complex task entirely and in a robust manner. In order to resolve this issue, two alternatives should be reviewed in future endeavors. First, based on the knowledge acquired during this work, new behavioral fitness metrics that would better guide the evolution of memory skills should be added to the multi-objective fitness function; for instance, it should be rewarded the emergence of linear output signals from the internal dynamics of a neurocontroller. For the second alternative, a further exploration of the search space should be performed by means of strategies such as novelty search, which rewards the originality of a solution.

On the other hand, upon further experimentation on evolved memory-endowed agents in modified environments and for varying simulation times, it was discovered that in order to solve the T-maze alternation task these agents rely on sensing the time and synchronizing their movements accordingly rather than on using spatial cues, either exclusively or in combination with temporal cues, i.e. they showed a temporal instead of a spatial or spatiotemporal working memory. In future work, the author intends to study the feasibility of evolving other types of and mechanisms for working memory by using the same neuroevolutionary approach.

This work also tested the competence of different neural architectures to foster the evolution of working memory behaviors. The experimental results showed that some neural models are inherently more advantageous than others to the evolution of memory skills. Further analyses revealed that in this particular experiment a temporal working memory arises from time-synchronized neural signals (for instance, a ramp-like signal), whose likelihood of emergence from an evolutionary perspective varies across different neural architectures depending on their intrinsic internal dynamics, and more specifically on their activation functions and network topologies. Thus, ultimately, the reason behind the varying results observed within the tested neural architectures is that some of them require a more precise tuning than others of their variable parameters, such as synaptic weights, time constants, etc., in order to produce nearly linear signals. In the end, the CTRNN and ESN+α models appear as the more favorable to the evolution of a working memory, although the latter could be further enhanced by applying an incremental two-stage evolutionary approach, under which the goal of the first evolutionary stage would be to generate nearly linear signals from the reservoir layer. In the case of the SNN+STDP and SNN architectures, due to the complexity of their neural workings, the questions of why these neural models were and were not, respectively, propitious for the development of working memory behaviors, and of how the Spike Response Model used could be modified in order to make it more promising to the evolution of memory skills, are left to be answered in further research work.

Lastly, most evolutionary runs took days while others, namely the ones associated with the SNN and SNN+STDP models, took weeks to complete on a personal computer. Therefore, with the aim of reducing this experimentation time, further studies should be conducted on a different hardware framework; one that exploits the intrinsic parallelism of artificial evolution. To this end, a cloud computing framework is regarded by the author to be the most promising approach.

References

[1] S. Doncieux, J.B. Mouret, N. Bredeche, V. Padois. Evolutionary Robotics: Exploring New Horizons. Springer Series: Studies in Computational Intelligence, New Horizons in Evolutionary Robotics, Springer, pp. 3-25. (2011).

[2] I. Harvey, E.A. Di Paolo, E. Tuci, R. Wood, M. Quinn. Evolutionary Robotics: A new scientific tool for studying cognition. Artificial Life, 11(1-2):79-98. (2005).

[3] P. Husbands, R. Moioli, Y. Shim, A. Philippides, P. Vargas, M. O'Shea. Evolutionary Robotics and Neuroscience. The Horizons of Evolutionary Robotics, MIT Press. (2013).

[4] E. Izquierdo, E.A. Di Paolo. Is an Embodied System ever purely Reactive? Proceedings

of the 8th European Conference of Artificial Life, ed. M.S. Capcarrere, Springer, pp. 252-261. (2005).

[5] S. Nolfi, D. Marocco. Evolving Robots able to integrate Sensory-Motor Information over Time. Theory in Biosciences, vol. 120, Urban & Fischer Verlag, pp. 287-310. (2001).

[6] N. Jakobi. Evolutionary Robotics and the Radical Envelope of Noise Hypothesis. (1997).

[7] E.A. Di Paolo, I. Harvey. Decisions and Noise: The Scope of Evolutionary Synthesis and Dynamical Analysis. (2003).

[8] G. Dudek, M. Jenkin. Locomotion: Differential Drive, Computational Principles of Mobile Robotics. 2nd edition, Cambridge University Press, pp. 39-41. (2010).

[9] H. Jaeger. The Echo State Approach to Analysing and Training Recurrent Neural Networks. (2010).

[10] W. Gerstner, J.L. van Hemmen, J.D. Cowan. What Matters in Neuronal Locking? Neural Computation, November 15, Vol. 8, No. 8, pp. 1653-1676. (1996).

[11] D. Floreano, C. Mattiussi. Evolution of Spiking Neural Controllers for Autonomous Vision-Based Robots. Evolutionary Robotics: From Intelligent Robotics to Artificial Life, Volume 2217 of the series Lecture Notes in Computer Science, pp. 38-61. (2001).

[12] E.A. Di Paolo. Evolving spike-timing-dependent plasticity for single-trial learning in robots. (2003).

[13] R.V. Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. Neural Computation 19 (6), pp. 1468-1502. (2007).

[14] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. (2002).

[15] D. Gupta, S. Ghafir. An Overview of Methods maintaining Diversity in Genetic Algorithms. International Journal of Emerging Technology and Advanced Engineering, Volume 2, Issue 5. (2012).

[16] J. Lehman, K.O. Stanley: Abandoning objectives: Evolution through the Search of Novelty Alone. Evolutionary Computation Journal, (19):2, pp. 189-223. (2011).