

An experimental comparison of evolved neural network models for controlling simulated modular soft robots



Giorgia Nadizar^a, Eric Medvet^{b,*}, Stefano Nichele^{c,d}, Sidney Pontes-Filho^{d,e,f}

^a DMG, University of Trieste, Trieste, Italy

^b DIA, University of Trieste, Trieste, Italy

^c Østfold University College, Halden, Norway

^d Oslo Metropolitan University, Oslo, Norway

^e Simula Research Laboratory, Oslo, Norway

^f Norwegian University of Science and Technology, Trondheim, Norway

ARTICLE INFO

Article history:

Received 1 February 2023

Received in revised form 12 June 2023

Accepted 2 July 2023

Available online 8 July 2023

Keywords:

Voxel-based soft robots

Neuroevolution

Spiking neural networks

ABSTRACT

Voxel-based soft robots (VSRs) are a type of modular robots composed by interconnected soft and deformable blocks, i.e., voxels. Thanks to the softness of their bodies, VSRs may exhibit rich dynamic behaviors. One open question is what type of neural controller is most suitable for a given morphology and sensory apparatus in a given environment. One observation is that artificial neural networks with state may be able to cope with the dynamical nature of VSR bodies and their morphological computation. In this work, we consider four types of controllers, i.e., multilayer perceptrons (MLPs, stateless), recurrent neural networks (RNNs), spiking neural networks (SNNs) without homeostasis, and SNNs with homeostasis. We consider three robot morphologies tested for locomotion, where each morphology is investigated in simulation with three different types and number of sensors. Neural network controllers are optimized with neuroevolution, and the experimental results are compared in terms of effectiveness, efficiency, and generalization ability. In addition, we analyze the resulting behavior of the robots systematically. Our results show that RNNs are typically more effective while MLPs are often the weakest controllers, particularly for robots with few sensors. However, SNNs are more capable in terms of generalization and the mechanism of homeostasis is often beneficial. Finally, we show that RNNs and SNNs with homeostasis produce a more wide variety of behaviors.

© 2023 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

1. Introduction

Artificial neural networks (ANNs) have been broadly and successfully used for a wide variety of computations ranging from language conversation [1] to playing diplomatic board games [2], including as controllers that generate robot commands on the fly [3].

Of particular interest for problems where the temporal dynamics are important, recurrent neural networks (RNNs) are one of the most interesting class of ANNs with state. Possessing a state encompasses a memory capacity or, in other words, the possibility of having cycles allowing outputs from a neuron to affect subsequent computations of the same node. Therefore RNNs can be considered as dynamical systems, making them a potential good fit as controllers for embodied agents acting in real time in response to their environment and the agent's previous actions.

However, in the domain of modular robotics, there is a lack of understanding of whether it is beneficial to implement recurrent neural controllers or spiking neural controllers for different morphologies and tasks.

In order to reduce such research gap, in this work, we consider four ANN variants, of which three of them with a state, and perform an extensive comparison when used as controllers for a type of simulated embodied agents: voxel-based soft robots (VSRs) [4]. VSRs are robots of particular interest in this context, since their body is composed of a collection of interconnected soft modules, hence capable of exhibiting a rich variety of dynamic behaviors [5].

Such dynamic processes conducted and enabled by the robot bodies are typically known as morphological computations [6]. In order to match the dynamics of the body and exploit them at their full potential, we investigate different types of evolved neural network models—that is, different ANN models for which we rely on evolution for the optimization of the parameters. In particular, we consider simple and stateless multilayer perceptrons (MLPs), recurrent neural networks, spiking neural networks (SNNs), the latter with and without homeostasis. The four named variants

* Corresponding author.

E-mail addresses: giorgia.nadizar@phd.units.it (G. Nadizar), emedvet@units.it (E. Medvet), stefano.nichele@hiof.no (S. Nichele), sidney@simula.no (S. Pontes-Filho).

differ in both their parameter size and state size; the former may be seen as a proxy for information processing expressiveness, i.e., how diverse may the processing doable by the ANN be; while the latter may be seen as a proxy for memory size. Homeostasis is a biological mechanism that restores a balance in a system behavior when abrupt changes in the input stimuli arise. In case of SNNs, such homeostatic plasticity acts as a regulator of a neuron firing rate by adjusting the threshold of the membrane potential.

Based on the hypothesis that recurrent and spiking neural models may be beneficial in the domain of VSRs, we conduct a large experimental campaign (in simulation) by considering nine robots differing in shape (biped, worm, and “comb”, a form of multiped) and sensory equipment (few or many, different types such as sight and touch) optimized by neuroevolution. The evolved neural networks are compared on the robots for the task of locomotion, i.e., the ability to move as far as possible from the initial position. The metrics used for comparison encompass the search effectiveness measured in term of velocity of the evolved robots, search efficiency in terms of the evolutionary time needed to obtain the best robot, and generalization ability tested as the evolved robot velocity in unknown conditions. Additionally, we perform a systematic investigation of the obtained robot behaviors using a variety of metrics further visualized using a principal component analysis for dimensionality reduction. Our experimental results show that indeed neural networks with state are particularly well suited for the control of voxel-based soft robots.

The main contributions and findings are the following: (1) RNNs are the most effective models while MLPs are on average the weakest network controllers for robots with few sensors. (2) SNNs appear to be more capable of generalization than other types of controllers. One interesting observation is that the mechanism of homeostasis is beneficial across a wide variety of scenarios, indeed providing an additional level of fast adaptation during the lifetime of the robotic agents. (3) ANNs with state such as RNNs and SNNs with homeostasis are able to produce a wide repertoire of diverse behaviors, which may be argued to be a beneficial property for robots to be deployed in unknown or changing environments [7].

The remainder of the paper is organized as follows. In Section 2, we survey the previous works that are relevant to our study. In Section 3, we present the three kinds of ANNs that we considered in our comparison under the same theoretical framework. In Section 4, we give an overview on VSRs, defining their morphology, sensory apparatus, and controller, the latter in an abstract way. In Section 5, we show how to use ANNs as controllers of a VSR. In Section 7, we describe our experimental comparison in detail and discuss the results under several points of view. Finally, in Section 9, we draw the conclusions.

2. Related work

Controlling a robot to achieve a certain task is a non-trivial problem, which can be tackled in different ways, ranging from classical control theory to more innovative methods based on artificial intelligence. A commonly used strategy involves equipping the robot with a *brain*, often in the form of an ANN [8,9], which is deputy to performing computation to drive the actions of the agent. A rather contrasting paradigm is that of morphological computation, according to which the *body* is itself responsible for performing computation, with the brain becoming secondary if not useless at all [6]. The embodied cognition paradigm [10] tries to bridge these antipodal views by localizing intelligence, i.e., the

source of computation, in the body–brain entanglement, for both real- and artificial-life agents.

Clearly, this paradigm does not apply equally well throughout the entire robotics domain, as some robot morphologies are naturally more suited for hosting computation than others. Intuitively, a richer body dynamics, as in soft robots [11], relates to a larger share of computational power residing in the body of the agent [12]. VSRs, the modular soft robots we employ in this work, naturally fit this scheme, and hence constitute a relevant test bed for assessing the body–brain interplay in a controlled simulated environment. Moreover, they are particularly well-suited for studying how real-world phenomena can be ported in the artificial domain, as, e.g., development [13,14] or morphological regeneration [15].

A similar argument can be made about the robotic controller: a given type of brain, i.e., ANN, may or may not be appropriate for a certain robot and/or task. In fact, in the variety of existing ANN models, one of the prime discerning factors is the structural support they offer for achieving computation in synergy with the body dynamics. In other words, the ANN choice is often driven by features such as having a state, i.e., a form of memory [16], being able to achieve self-regulation and/or unsupervised learning [17,18], or be tailored to specific kinds of inputs (e.g., high-dimensional visual description of the environment). While a controller with memory may be more suitable to cope with complex tasks that require a deeper cognition of the environment by the robot, the complex dynamics induced by a controller with state may interfere with the dynamics of the robot body—this being a key motivation behind our study. On the other hand, learning (or similar forms of adaptation) are often desirable, in particular when the controller is not the only part of the robot being subjected to optimization [19]; however, the overhead of complexity needed to allow for auto-adaptation may sometimes hinder the robot effectiveness. The choice of a given neural model is hence not straightforward, and the need arises for comparing different options. Jin et al. [20] surveyed the topic for a robot manipulation task, encompassing a wide range of ANNs, including most of the ones we experiment with in our study, but also Echo State Networks, Dual Networks, and Central Pattern Generators. Similarly to said study, here we aim at comparing different ANN models for a single task, yet, differently from the cited work we consider (simulated) modular robots tackling the task of locomotion and we conduct an experimental evaluation rather than a survey.

The most commonly used model of ANN is the multilayer perceptron [21,22], which is the usual go-to regardless of the robot type and the application domain. Such ANNs have already been successfully used for VSRs both as centralized controllers and as distributed ones to take advantage of the robot modularity [7]. To achieve higher biological resemblance and to increase the computational efficiency, Nadizar et al. [23] introduced the practice of pruning to MLPs in VSRs, whereas Ferigo et al. [24] introduced unsupervised Hebbian learning to MLPs in VSRs with the aim of improving their performance and generalization ability.

Recurrent neural networks are another often used ANN model in robot control [25,26]. These ANNs share the simplicity of MLPs, yet possess a state which makes them ideal for tasks requiring a form of memory, as, e.g., navigation [27,28]. These ANNs have never been applied to VSRs per se, although the distributed controller proposed by Pigozzi et al. [7] exploits recurrence in the information processing within the robot.

Unlike MLPs and RNNs, spiking neural networks have only recently started gaining popularity in the robotics domain, even though the first pioneering work involving them for controlling modular robots dates back to 2003 [29]. Despite SNNs were primarily introduced aiming at biological resemblance [30,31],

nowadays they do not only attract interest from the biological/neuroscientific perspective. In fact, these ANNs possess unique features also from the computational perspective, as they allow for homeostasis [32]—a form of aparametric unsupervised learning—and are highly energy efficient in combination with neuromorphic hardware [33]. Bing et al. [34] provide an overview of the applications of SNNs in robotics. The ones that are most relevant for this study are the works by Spaeth et al. [33,35], which involve a form of modular soft robots analogous to VSRs and the locomotion task. Such works use an SNN as a central pattern generator [36,37] that controls the contraction and expansion of simulated robot muscles similarly to the voxels in the VSRs. However, the network of a central pattern generator is manually designed and consists of inhibitory cycles between groups of neurons for creating a rhythmic activation of the muscles [38]. In addition, Nadizar et al. [39] applied SNNs to VSRs in a distributed manner, then each voxel is controlled by a single network that can communicate with its neighboring voxels network. The goal of this method is the achievement of a form of collective intelligence among the single voxels. SNNs can create cognitive maps in space or time dimensions that can work as a form of spatial memory, hence tackling tasks different than locomotion, as, e.g., robot path planning and state predictions [40–42]. SNNs can also be used to make robots to learn associations, for example between objects and actions. This is called associative learning [43–45].

For each of these ANNs to become effective robot controllers, a training, i.e., an optimization, phase is needed. Given the diverse models involved, we rely on neuroevolution for the optimization, as it can be applied equally well to all the considered ANNs. In particular, unlike reinforcement learning, neuroevolution does not require specific domain-knowledge in crafting the reward (as in [46]) and can thus easily be applied to a broad range of tasks. We refer the reader to [47] for an overview of different neuroevolution-based methodologies for optimizing ANN-based controllers for the navigation task.

3. Background: neural models

ANNs are computational models that take inspiration from biological neural networks. ANNs are composed of artificial neurons interconnected by weighted connections that resemble the different intensities of synaptic contact. Neurons are modeled as electrically excitable nerve cells which pass electric signals along the synapses [48].

ANNs attempt to mimic biological neural circuits in order to reflect their behavior and adaptive features. Therefore, they are often used for solving difficult problems, e.g., modeling complex relationships between inputs and outputs [49]. In this study, we exploit ANNs as robotic controllers, so we aim at inferring a relationship between the state and the perceptions of the robot, and its actions.

Notably, since we deploy ANNs to control agents that “live” in time, we are implicitly introducing the notion of time within the ANNs. Namely, we simulate the ANN evolution over time in a discrete manner, with a time resolution Δt_h . Remarkably, Δt_h does not necessarily need to match the time resolution of the physics simulation of the robot body, hence the ANN can be updated at different instants with respect to the robot simulation. Due to the stateful nature of some neural models employed, the ANNs can be considered here as dynamical systems, for which the outputs depend not only on the current inputs but also on the previous history of the system, reflected by the ANN *state*. We can hence describe an ANN using a notation borrowed from dynamical systems:

$$\mathbf{s}^{(h)} = g(\mathbf{x}^{(h)}, \mathbf{s}^{(h-1)}), \quad (1)$$

$$\mathbf{y}^{(h)} = f(\mathbf{x}^{(h)}, \mathbf{s}^{(h)}), \quad (2)$$

$$\mathbf{s}^{(0)} = \mathbf{s}_0, \quad (3)$$

where $\mathbf{s}^{(h)} \in \mathbb{R}^{|\mathcal{S}|}$ is the state of the ANN at time step h , $\mathbf{x}^{(h)} \in \mathbb{R}^m$ is the input at h , $\mathbf{y}^{(h)} \in \mathbb{R}^n$ is the output at h , $g: \mathbb{R}^m \times \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}$ is the function determining the state update given an input, $f: \mathbb{R}^m \times \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^n$ is the function determining the output given the current state and input, and \mathbf{s}_0 is the initial state.

Since the very first computational model proposed by McCulloch and Pitts [21] in 1943 – the *perceptron* – a large variety of ANN models have originated, differing along various axes, e.g., how neurons are modeled or which architectures are employed, and covering a vast set of features, such as simplicity, computational efficiency, or biological resemblance. Here, we take into consideration a subset of models, aimed at covering different aspects, as we shall see in the following sections.

In detail, we consider MLPs, RNNs, and SNNs. While the two first neural models have been extensively used for robot control in the last decades [8], SNNs have started being used more recently. We focus on these three models, among the many that are applicable to the task of robot control [20], because we believe they exhibit different trade-offs between their parameter size and state size. We see the former as a proxy for information processing expressiveness, i.e., how diverse may the processing doable by the ANN be, and the latter as a proxy for memory size. Moreover, the parameter size directly influences the search space of the (evolutionary) optimization process: in principle, the greater the number of parameter describing an ANN, the longer the search for suitable parameter values. Finally, we remark that more complex ANN models that have been used for controlling robotic agents, such as long short-term memory ANNs [50], are often deployed in a controller with a large input space, e.g., resulting from vision sensors. In contrast, the robots considered in our scenario are equipped with simple sensors that only generate low-dimension inputs (see Section 4).

3.1. Multilayer perceptrons

The first model we consider is the multilayer perceptron, that is a fully connected feed forward ANN, where each neuron is a perceptron [21,22] and neurons are organized in layers. The MLP has no state, which means that, in terms of Eqs. (1)–(3), $|\mathcal{S}| = 0$, g and \mathbf{s}_0 are undefined, and f takes only $\mathbf{x}^{(h)}$ as argument.

The processing of $\mathbf{y}^{(h)} = f(\mathbf{x}^{(h)})$ in an MLP is based on the processing occurring in a single neuron:

$$v_{l,i}^{(h)} = \varphi^P \left(\left(\sum_{j=1}^{m_{l-1}} w_{l,i,j} v_{l-1,j}^{(h)} \right) - b_{l,i} \right), \quad (4)$$

where $\varphi^P: \mathbb{R} \rightarrow \mathbb{R}$ is the *activation function* of the perceptron, $v_{l,i}$ is the activation level of the i th neuron in the l th layer, $w_{l,i,j}$ is the synaptic weight associated with the synapse connecting the i th neuron in the l th layer with the j th neuron in the $l-1$ th layer, $b_{l,i}$ is the bias of the i th neuron in the l th layer, and m_l is the size, i.e., the number of neurons in the l th layer. Since MLPs are organized into layers, respectively one input layer, zero or more hidden layers, and one output layer, the computation described by Eq. (4) is performed layer by layer, meaning that the outputs of the previous layer become the inputs of the following one, for each time step h . Eq. (4) can be written more concisely by replacing the summation with vector product:

$$v_{l,i}^{(h)} = \varphi^P \left(\mathbf{w}_{l,i}^T \mathbf{v}_{l-1}^{(h)} - b_{l,i} \right). \quad (5)$$

Hence, for the l th layer:

$$\begin{aligned} \mathbf{v}_l^{(h)} &= \begin{bmatrix} \varphi^P \left(\mathbf{w}_{l,1}^\top \mathbf{v}_{l-1}^{(h)} - b_{l,1} \right) \\ \vdots \\ \varphi^P \left(\mathbf{w}_{l,m_l}^\top \mathbf{v}_{l-1}^{(h)} - b_{l,m_l} \right) \end{bmatrix}^\top \\ &= \varphi_{\theta_l}^{\text{LP}} \left(\mathbf{v}_{l-1}^{(h)} \right), \end{aligned} \quad (6)$$

where $\theta_l \in \mathbb{R}^{(m_{l-1}+1)m_l}$ is a numerical vector containing all the synaptic weights and biases and parameterizing the function $\varphi_{\theta_l}^{\text{LP}} : \mathbb{R}^{m_{l-1}} \rightarrow \mathbb{R}^{m_l}$ that describe the processing for one layer of neurons. Finally, the entire MLP processing can be completely described in terms of Eq. (2) by:

$$\begin{aligned} \mathbf{y}^{(h)} &= \mathbf{v}_{l^*+1}^{(h)} = \varphi_{\theta_{l^*+1}}^{\text{LP}} \left(\varphi_{\theta_{l^*}}^{\text{LP}} \left(\dots \left(\varphi_{\theta_1}^{\text{LP}} \left(\mathbf{v}_0^{(h)} \right) \right) \right) \right) \\ &= f_{\theta}^{\text{MLP}} \left(\mathbf{v}_0^{(h)} \right) = f_{\theta}^{\text{MLP}} \left(\mathbf{x}^{(h)} \right), \end{aligned} \quad (7)$$

where $\mathbf{v}_0^{(h)} \triangleq \mathbf{x}^{(h)}$, l^* is the number of hidden layers in the MLP, with $l = 0$ corresponding to the input layer (for which $m_0 = m$) and $l = l^* + 1$ corresponding to the output layer (for which $m_{l^*+1} = n$, and $\theta \in \mathbb{R}^p$ is a numerical vector containing all the parameters of f_{MLP} , with $p = \sum_{l=1}^{l^*+1} |\theta_l| = \sum_{l=1}^{l^*+1} (m_{l-1} + 1)m_l$.

Summarizing, an MLP with l^* hidden layers has a state of size $|\mathbf{s}| = 0$ and can be described with a parameter vector of size $|\theta| = \sum_{l=1}^{l^*+1} (m_{l-1} + 1)m_l$. Moreover, the MLP processing depends on one single network-wise parameter, the activation function φ^P .

3.2. Recurrent neural networks

To include memory, we consider another neural model: recurrent neural networks, with perceptrons as elementary units. In this case, the output of each unit is computed with φ^P , neurons are organized in layers, as for MLPs, but the way neurons are connected is different. In particular, the topology we opt for consists of an input, an output layer, which are built similarly to those of MLPs, and l^* recurrent layers in between them. The input synapses fully connect neurons of the input layer with those of the first recurrent layer, while the output synapses fully connect the last recurrent layer with the output layer. In each recurrent layer, every neuron is connected with every other neuron in the same layer (also considering auto-synapses) and every neuron in the previous layer. In this case, to avoid infinite recursion, the inputs of the recurrent neurons at time h are the outputs of the same layer at the previous time step $h - 1$, together with those computed by the input layer at time h .

In mathematical terms this becomes, for the single neuron in the l th recurrent layer:

$$v_{l,i}^{(h)} = \varphi^P \left(\mathbf{w}_{l,i}^\top \mathbf{v}_{l-1}^{(h)} + \mathbf{w}'_{l,i}{}^\top \mathbf{v}_l^{(h-1)} - b_{l,i} \right), \quad (8)$$

where $\mathbf{w}'_{l,i} \in \mathbb{R}^{m_l^2}$ is the vector of the synaptic weights for the recurrent synapses, i.e., those connecting the recurrent layer neurons with each other. More concisely, for the entire recurrent l th layer:

$$\begin{aligned} \mathbf{v}_l^{(h)} &= \begin{bmatrix} \varphi^P \left(\mathbf{w}_{l,1}^\top \mathbf{v}_{l-1}^{(h)} + \mathbf{w}'_{l,1}{}^\top \mathbf{v}_l^{(h-1)} - b_{l,1} \right) \\ \vdots \\ \varphi^P \left(\mathbf{w}_{l,m_l}^\top \mathbf{v}_{l-1}^{(h)} + \mathbf{w}'_{l,m_l}{}^\top \mathbf{v}_l^{(h-1)} - b_{l,m_l} \right) \end{bmatrix}^\top \\ &= \varphi_{\theta_l}^{\text{RP}} \left(\mathbf{v}_{l-1}^{(h)}, \mathbf{v}_l^{(h-1)} \right), \end{aligned} \quad (9)$$

with $\theta_l \in \mathbb{R}^{(m_{l-1}+1)m_l+m_l^2}$ and $\varphi_{\theta_l}^{\text{RP}} : \mathbb{R}^{m_{l-1}} \times \mathbb{R}^{m_l} \rightarrow \mathbb{R}^{m_l}$.

In terms of Eqs. (1) and (2), and by associating the state $\mathbf{s} = [\mathbf{v}_1^{(h)} \dots \mathbf{v}_{l^*}^{(h)}] \in \mathbb{R}^{\sum_l m_l}$ with the activation level of the recurrent layers, we can write:

$$\mathbf{s}^{(h)} = g_{\theta}^{\text{RNN}} \left(\mathbf{v}_0^{(h)}, \mathbf{s}^{(h-1)} \right) = g_{\theta}^{\text{RNN}} \left(\mathbf{x}^{(h)}, \mathbf{s}^{(h-1)} \right), \quad (10)$$

$$\mathbf{y}^{(h)} = \varphi_{\theta_{l^*+1}}^{\text{LP}} \left(\mathbf{v}_{l^*}^{(h)} \right) = f_{\theta}^{\text{RNN}} \left(\mathbf{s}^{(h)} \right), \quad (11)$$

with $\mathbf{v}_0^{(h)} \triangleq \mathbf{x}^{(h)}$.

From these equations the statefulness of RNNs becomes glaring, as opposed to the stateless MLPs. In fact, here the computation at time step h directly and explicitly depends from the results obtained at $h - 1$. This is a form of memory of the ANN, which can keep track of previous events and neural activity, and eventually re-use the information stored for future computations.

Summarizing, an RNN has a state with size $|\mathbf{s}| = \sum_{l=1}^{l^*} m_l$ and can be described with a parameter vector with size $|\theta| = \sum_{l=1}^{l^*+1} |\theta_l| = (m_{l^*} + 1)m_{l^*+1} + \sum_{l=1}^{l^*} (m_{l-1} + 1)m_l + m_l^2$ (the first part is different because the output layer, i.e., the $l^* + 1$ th one, is not recurrent). Moreover, like for the MLP, the RNN processing depends only on the single network-wise parameter φ^P .

3.3. Spiking neural networks

The two aforementioned neural models, MLPs and RNNs, are both simple and fairly computationally efficient, yet they lack in terms of biological plausibility. Hence, we move towards spiking neural networks for our third model, for which biological resemblance plays a fundamental role [30]. The key element of SNNs is the modeling of the evolution over time of the membrane potential of neurons, which can be modified by incoming excitatory or inhibitory neural stimuli, occurring in the form of spikes that propagate along synapses. The generation of said spikes is called *firing*, and arises whenever the membrane potential of a neuron exceeds a given threshold. Despite the binary nature of spikes, the intensity of any stimulus received by a neuron is modulated by the strength of the synapse connecting the firing neuron (pre-synaptic neuron) and the neuron receiving the spike (post-synaptic neuron), similarly to what we have seen in Sections 3.1 and 3.2.

The core difference between SNNs and MLPs or RNNs hence lies in the way information is encoded: in SNNs information is embedded in the distribution of spikes over time, whereas MLPs and RNNs encode information as real values at every instant. Therefore, to allow the use of SNNs in a framework designed to be coupled with canonical ANNs, we require additional procedures deputy to converting the spike trains to real values and vice versa, i.e., to making the SNN consistent with Eqs. (1) and (2), where the input is a numerical vector $\mathbf{x}^{(h)} \in \mathbb{R}^m$ and the output is a numerical vector $\mathbf{y}^{(h)} \in \mathbb{R}^n$. We describe those conversion procedures in Section 5.2.

Within the SNNs paradigm, several neuron models exist [31], which all share the main concepts derived from neuroscience. Among them, we employ the computationally efficient leaky integrate and fire (LIF) model, simulated in discrete time. The LIF model represents the neural membrane as a capacitor, the potential of which can be increased or decreased by inputs (excitatory or inhibitory), and exponentially decays with time.

Concerning the architecture of the SNN, we keep a fully connected feed forward layered topology, similarly to an MLP, with each neuron being an LIF neuron instead of a perceptron. Hence, at each neural simulation time step h , the membrane potential

$v_{l,i}^{(h)}$ of the i th LIF neuron of the l th layer is:

$$\begin{aligned} v_{l,i}^{(h)} &= v_{l,i}^{(h-1)} - \Delta t_h \lambda_v v_{l,i}^{(h-1)} + \sum_{j=1}^{j=m_{l-1}} w_{l,i,j} v_{l-1,j}^{(h)} \\ &= v_{l,i}^{(h-1)} (1 - \Delta t_h \lambda_v) + \mathbf{w}_{l,i} \mathbf{v}_{l-1}^{(h)} \\ &= \varphi_{\mathbf{w}_{l,i}}^M \left(\mathbf{v}_{l-1}^{(h)}, v_{l,i}^{(h-1)} \right), \end{aligned} \quad (12)$$

with $v_j^{(h)} \in \{0, 1\}$ carrying the pre-synaptic neuron spike. After the update, if the membrane potential $v_{l,i}^{(h)}$ exceeds a threshold $\vartheta_{l,i}^{(h)}$, the neuron outputs a spike, i.e., $v_{l,i}^{(h)} = 1$, and the membrane potential is reset to its resting value v_{rest} , otherwise $v_{l,i}^{(h)} = 0$. Formally:

$$\begin{aligned} v_{l,i}^{(h)} &= \begin{cases} \varphi_{\mathbf{w}_{l,i}}^M \left(\mathbf{v}_{l-1}^{(h)}, v_{l,i}^{(h-1)} \right) & \text{if } \varphi_{\mathbf{w}_{l,i}}^M \left(\mathbf{v}_{l-1}^{(h)}, v_{l,i}^{(h-1)} \right) \leq \vartheta_{l,i}^{(h)} \\ v_{\text{rest}} & \text{otherwise} \end{cases} \\ &= \varphi_{\mathbf{w}_{l,i}}^{v\text{LIF}} \left(\mathbf{v}_{l-1}^{(h)}, \vartheta_{l,i}^{(h)}, v_{l,i}^{(h-1)} \right), \end{aligned} \quad (13)$$

$$\begin{aligned} \vartheta_{l,i}^{(h)} &= \begin{cases} 0 & \text{if } \varphi_{\mathbf{w}_{l,i}}^M \left(\mathbf{v}_{l-1}^{(h)}, v_{l,i}^{(h-1)} \right) \leq \vartheta_{l,i}^{(h)} \\ 1 & \text{otherwise} \end{cases} \\ &= \varphi_{\mathbf{w}_{l,i}}^{v\text{LIF}} \left(\mathbf{v}_{l-1}^{(h)}, \vartheta_{l,i}^{(h)}, v_{l,i}^{(h-1)} \right), \end{aligned} \quad (14)$$

which can be ported to the corresponding layer-wise form, where $\varphi_{\theta_l}^{v\text{LIF}} : \{0, 1\}^{m_{l-1}} \times \mathbb{R}^{m_l} \times \mathbb{R}^{m_l} \rightarrow \mathbb{R}^{m_l}$ and $\varphi_{\theta_l}^{v\text{LIF}} : \{0, 1\}^{m_{l-1}} \times \mathbb{R}^{m_l} \times \mathbb{R}^{m_l} \rightarrow \{0, 1\}^{m_l}$, where $\theta_l = [\mathbf{w}_{l,1} \dots \mathbf{w}_{l,m_l}] \in \mathbb{R}^{m_{l-1} \times m_l}$ is, similarly to the MLP, a numerical vector containing all the synaptic weights (with no biases, though) for the l th layer.

We enhance the LIF model introducing the biological concept of homeostatic plasticity. Homeostasis is a self regulatory mechanism present at various sites of living organisms, which aims at re-establishing equilibrium in contrast to strong stimuli that could unbalance a system [32]. In our case, homeostasis operates as a firing rate regulator, acting on the threshold $\vartheta_{l,i}^{(h)}$ of neurons, to prevent excessive or too scarce activity:

$$\begin{aligned} \vartheta_{l,i}^{(h)} &= \min \left(\vartheta_{l,i}^{(h-1)}, \sum_{j=1}^{j=m_l} w_{l,i,j} \right) + \psi_{l,i}^{(h)} \\ &= \varphi_{\mathbf{w}_{l,i}}^{\vartheta\text{LIF}} \left(\psi_{l,i}^{(h)}, \vartheta_{l,i}^{(h-1)} \right), \end{aligned} \quad (15)$$

with $\psi_{l,i}^{(h)}$ being a parameter updated as:

$$\begin{aligned} \psi_{l,i}^{(h)} &= \begin{cases} \psi_{l,i}^{(h-1)} + \psi_{\text{inc}} & \text{if } v_{l,i}^{(h-1)} = 1 \\ \psi_{l,i}^{(h-1)} - \psi_{l,i}^{(h-1)} \lambda_\psi \Delta t_h & \text{otherwise.} \end{cases} \\ &= \varphi_{\psi_{l,i}}^{\psi\text{LIF}} \left(v_{l,i}^{(h-1)}, \psi_{l,i}^{(h-1)} \right). \end{aligned} \quad (16)$$

Both the equations describing homeostasis can be ported to the corresponding layer-wise form, where $\varphi_{\theta_l}^{\vartheta\text{LIF}} : \mathbb{R}^{m_l} \times \mathbb{R}^{m_l} \rightarrow \mathbb{R}^{m_l}$ and $\varphi_{\theta_l}^{\psi\text{LIF}} : \{0, 1\}^{m_l} \times \mathbb{R}^{m_l} \rightarrow \mathbb{R}^{m_l}$.

Overall, the processing of the l th layer of an SNN is described by:

$$\mathbf{v}_l^{(h)} = \varphi_{\theta_l}^{v\text{LIF}} \left(\mathbf{v}_{l-1}^{(h)}, \boldsymbol{\psi}_l^{(h-1)} \right), \quad (17)$$

$$\boldsymbol{\vartheta}_l^{(h)} = \varphi_{\theta_l}^{\vartheta\text{LIF}} \left(\boldsymbol{\psi}_l^{(h)}, \boldsymbol{\vartheta}_l^{(h-1)} \right), \quad (18)$$

$$\mathbf{v}_l^{(h)} = \varphi_{\theta_l}^{v\text{LIF}} \left(\mathbf{v}_{l-1}^{(h)}, \boldsymbol{\vartheta}_l^{(h)}, \mathbf{v}_l^{(h-1)} \right), \quad (19)$$

$$\mathbf{v}_l^{(h)} = \varphi_{\theta_l}^{v\text{LIF}} \left(\mathbf{v}_{l-1}^{(h)}, \boldsymbol{\vartheta}_l^{(h)}, \mathbf{v}_l^{(h-1)} \right). \quad (20)$$

It can be seen that the state, for the l -layer, is given by $\mathbf{s}_l = [\mathbf{v}_l \ \boldsymbol{\vartheta}_l \ \boldsymbol{\psi}_l] \in \mathbb{R}^{3m_l}$, for which we can write:

$$\begin{aligned} \mathbf{s}_l^{(h)} &= \begin{bmatrix} \varphi_{\theta_l}^{v\text{LIF}} \left(\mathbf{v}_{l-1}^{(h)}, \boldsymbol{\vartheta}_l^{(h)}, \mathbf{v}_l^{(h-1)} \right) \\ \varphi_{\theta_l}^{\vartheta\text{LIF}} \left(\boldsymbol{\psi}_l^{(h)}, \boldsymbol{\vartheta}_l^{(h-1)} \right) \\ \varphi_{\psi_{l,i}}^{\psi\text{LIF}} \left(\mathbf{v}_l^{(h-1)}, \boldsymbol{\psi}_l^{(h-1)} \right) \end{bmatrix}^T \\ &= \varphi_{\theta_l}^{s\text{LIF}} \left(\mathbf{v}_{l-1}^{(h)}, \mathbf{s}_l^{(h-1)} \right). \end{aligned} \quad (21)$$

The processing of the entire SNN with l^* hidden layers may be described, in terms of Eqs. (1) and (2), as:

$$\begin{aligned} \mathbf{s}^{(h)} &= \mathcal{G}_{\theta}^{\text{SNN}} \left(\mathbf{v}_{-1}^{(h)}, \mathbf{s}^{(h-1)} \right) \\ &= \mathcal{G}_{\theta}^{\text{SNN}} \left(\mathbf{x}^{(h)}, \mathbf{s}^{(h-1)} \right), \end{aligned} \quad (22)$$

$$\mathbf{y}^{(h)} = \mathbf{v}_{l^*+1}^{(h)} = \mathcal{f}_{\theta}^{\text{SNN}} \left(\mathbf{s}^{(h)} \right), \quad (23)$$

with $\mathbf{v}_{-1}^{(h)} \triangleq \mathbf{x}^{(h)}$.

Note that, differently than MLPs and the RNNs, the inputs and outputs of SNNs are spikes, rather than real values, i.e., they are defined in $\{0, 1\}^m$ and $\{0, 1\}^n$ respectively. We describe in Section 5.2 how we transform (sequences of) spikes to real values.

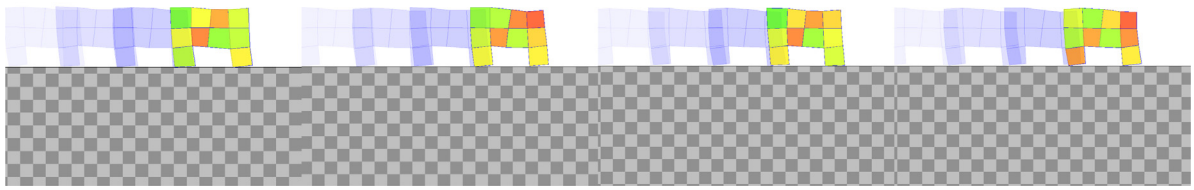
Summarizing, an SNN with l^* hidden layers and homeostasis, that we denote with SNN-H in the following, has a state with size $|\mathbf{s}| = \sum_{l=0}^{l=l^*+1} |\mathbf{s}_l| = 3 \sum_{l=0}^{l=l^*+1} m_l$ and can be described with a parameter vector with size $|\theta| = \sum_{l=1}^{l=l^*+1} |\theta_l| = \sum_{l=1}^{l=l^*+1} m_{l-1} m_l$. Moreover, the SNN-H processing depends on the following network-wise parameters: Δt_h , λ_v , v_{rest} , λ_ψ , and ψ_{inc} .

We also employ an SNN without homeostasis, that we denote with just SNN in the following. For this neural model, each $\vartheta_{l,i}^{(h)}$ is the same and remains constant, i.e., $\vartheta_{l,i}^{(h)} = \vartheta_0, \forall l, i, h$; hence, the state of the SNN for each l th layer does not include $\boldsymbol{\vartheta}_l$ and $\boldsymbol{\psi}_l$. As a consequence, an SNN with l^* hidden layers and without homeostasis has a state with size $|\mathbf{s}| = \sum_{l=0}^{l=l^*+1} |\mathbf{s}_l| = \sum_{l=0}^{l=l^*+1} m_l$ and can be described with a parameter vector with the same size of the SNN-H, i.e., $|\theta| = \sum_{l=1}^{l=l^*+1} m_{l-1} m_l$. Finally, the SNN processing depends on the following network-wise parameters: Δt_h , λ_v , v_{rest} , and ϑ_0 .

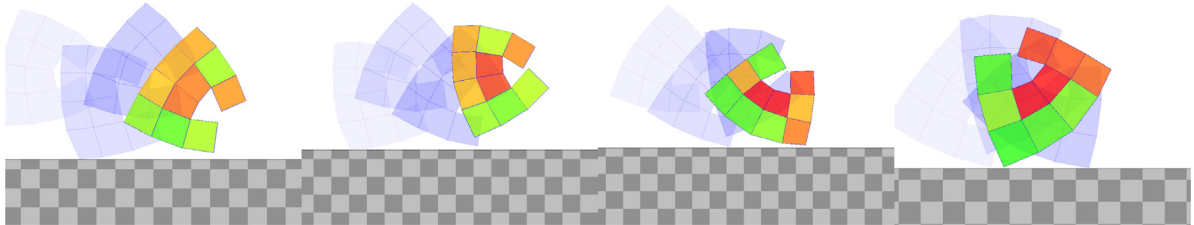
4. Background: voxel-based soft robots

Voxel-based soft robots are a kind of modular robots, composed of several soft elastic cubes, which are rigidly linked together in a predefined shape [4]. Each voxel changes its volume over time as a consequence of the joint effect of external and internal forces acting on it. Among the first ones we list gravity and the forces exerted by other bodies in contact with it, whereas the latter forces are those dictated by a *control signal*, determining the active contraction/expansion of the voxel. The movement of the robot is a consequence of the rhythmical and synergic action of voxels, together with their interaction with the environment. This gives rise to the robot behavior, which, if properly tuned, can lead to the accomplishment of several tasks: Fig. 1 shows four examples of a VSR doing locomotion, i.e., attempting to move the fastest possible along the positive x direction on a flat terrain. In this scenario, the success of the robot is determined by the combination of its morphology and its controller, which can both be tailored and optimized in a goal-specific manner. In this study, we handcraft morphologies, while we employ optimization in the form of neuroevolution for the controller fine-tuning.

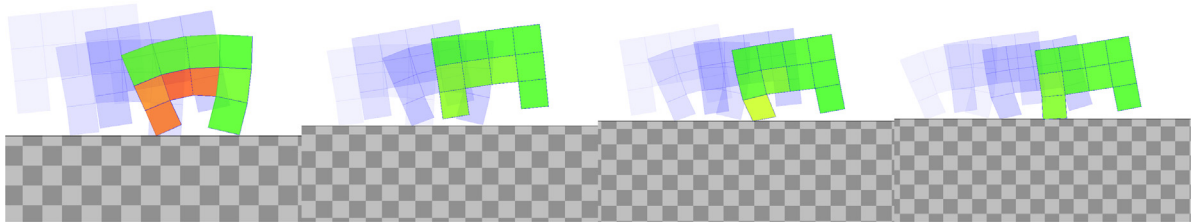
We perform our experimental evaluation on a 2D simulated variant of VSRs [51], simulated in continuous space and discrete time, with a time resolution for the physics simulation of $\Delta t_s = \frac{1}{60}$ s ($f_s = 60$ Hz). Hence, in this case VSRs become compounds



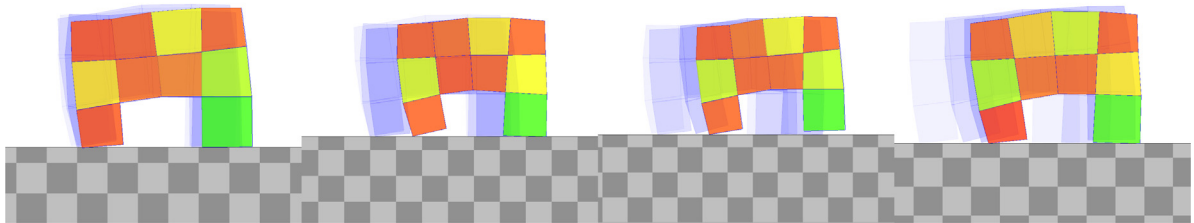
(a) Behavior A



(b) Behavior B



(c) Behavior C



(d) Behavior D

Fig. 1. Time-lapses showing four behaviors of a biped VSR doing locomotion—namely, the shown behaviors correspond to named markers of the biped panel of Fig. 8. Frames are taken at 1 s intervals (simulated time); in each frame, the blue shades show the position of the robots at earlier 0.5 s intervals. The color of each voxel represents the current area ratio with respect to the rest area (red for contracted, i.e., < 1 , green for expanded, i.e., > 1 , yellow otherwise, i.e., ≈ 1).

of elastic squares, whose area changes over time to ignite movement, instead of cubes with changing volume. However, we deem our results to be conceptually portable to the 3D case, and, eventually, to the real-world.

4.1. VSR morphology: shape and sensory apparatus

The morphology of a VSR describes the shape of its body and its sensory apparatus. The *shape* of a VSR is defined as a two-dimensional grid of voxels. This definition descends from the fact that all voxels are modeled as ideal squares, which can be easily arranged in a grid.

From the physical point of view, we model each voxel as a compound of many spring–damper systems linking four masses at the corners, as originally proposed by Medvet et al. [51]. The spring–damper systems endow the voxel with softness and

elasticity. The masses are the key for allowing the assembly of VSRs, as they can be welded to rigidly link together adjacent voxels at their corners. Without loss of generality, we model all voxels alike in terms of mechanical model parameters, meaning that all voxels share the same mechanical properties. We remark, though, that it is possible to easily change the body properties of VSRs by tweaking the parameters of the mechanical model [12].

Moving on to the *sensory apparatus* of VSR, we define it as the compound of sensors available to the VSR and their placement along the VSR body. Sensors can be used by the VSR for proprioception and to perceive various aspects of the surrounding environment. At each time step k , each j th sensor outputs a reading $r_j^{(k)} \in [0, 1]$. Here, we employ four types of sensors, which can be used to gather different kinds of information.

1. Area sensors sense the ratio between the current area of the voxel they are placed in and its rest area.

2. Touch sensors perceive if a voxel is in contact with the ground, $r_j^{(k)} = 1$, or not, $r_j^{(k)} = 0$.
3. Velocity sensors sense the velocity of the voxel center of mass along the voxel x - or y -axes.
4. Proximity sensors perceive the distance towards objects along a predefined direction α . The sensed value corresponds to the distance between the voxel center of mass and the closest object. In case no object is detected below a distance threshold d , the corresponding reading is set to d .

For area, velocity, and proximity sensors we employ a soft normalization of the outputs using the tanh function and rescaling, to ensure each output lies in $[0, 1]$.

To achieve movement, the voxels behave similarly to biological muscles, rhythmically contracting and expanding, as dictated by a control signal. Said signal is defined, for each i th voxel at each time step k , as $a_i^{(k)} \in [-1, 1]$: -1 corresponding to maximum requested contraction, and 1 corresponding to maximum requested expansion. Both expansion and contraction are modeled as instantaneous and linear variations of the springs within the simulated voxels. Concerning the computation of the control signal, we delve into the matter in the next section.

4.2. VSR controller: a neural controller

The controller of a VSR is responsible for computing, at each time step, the control signal which guides the voxels movement. Even though it is in principle possible to rely on a simple periodic function as a controller, constituting a trivial form of open-loop controller, here we resort to a more sophisticated closed-loop controller. In particular, we apply ANNs to process sensor readings, in order to compute the voxels control signals. Given the application of ANNs for the control task, we call this a *neural controller*.

Formally speaking, any closed-loop controller for a VSR can be considered as a dynamical system described by the two functions $g: \mathbb{R}^m \times \mathbb{R}^{|\mathbf{s}|} \rightarrow \mathbb{R}^{|\mathbf{s}|}$ and $f: \mathbb{R}^m \times \mathbb{R}^{|\mathbf{s}|} \rightarrow \mathbb{R}^n$ and by the initial state \mathbf{s}_0 (see Eqs. (1) to (3)), where m is the overall number of sensors deployed on the VSR, n is the number of voxels in the VSR, and $|\mathbf{s}|$ is the controller state dimension. In this work, we use as controllers the ANNs described in Section 5, that are all parameterized by a numerical vector $\theta \in \mathbb{R}^p$. The way the neural controller works can be hence described by:

$$\mathbf{s}^{(k)} = g_{\theta}(\mathbf{r}^{(k)}, \mathbf{s}^{(k-1)}), \quad (24)$$

$$\mathbf{a}^{(k)} = f_{\theta}(\mathbf{r}^{(k)}, \mathbf{s}^{(k)}), \quad (25)$$

$$\mathbf{s}^{(0)} = \mathbf{s}_0, \quad (26)$$

where $\mathbf{r}^{(k)}$ is the vector of the m sensor readings at time step k and $\mathbf{a}^{(k)}$ is the vector of the n control values at k .

It logically follows that as θ varies, the functions f_{θ} and g_{θ} can also change a lot, inducing extremely different behaviors in the VSR. Therefore, we aim at optimizing the parameters θ to achieve the desired behavior, as we shall see in Section 6.

An additional degree of freedom within the VSR controller is imposed by the control frequency, f_c , which regulates how often the controller is queried to compute new control signals for the voxels. The maximum control frequency corresponds to the simulation frequency $f_s = 60$ Hz, meaning that the controller is queried at every simulation time step. However, lower frequencies $f_c < 60$ Hz are also allowed: in those cases the controller is queried every $\lfloor \frac{f_s}{f_c} \rfloor$ simulation time steps, while the control signal stays unchanged in between two subsequent queries. Here we experiment both with $f_c = 60$ Hz and $f_c = 4$ Hz.

In the next section, we describe how we embed an instance of an ANN of one of the three considered models (MLP, RNN, and SNN) within a VSR, focusing in particular on the mapping between the ANN and the VSR working frequencies. We remark that, while the coupling of each of the three considered models to VSRs is not, per se, novel, this study is the first to provide a unified framework that describes in common terms these neural models as dynamical systems and shows how their evolution over time relates to the evolution of the embedding system, i.e., the robot.

5. Embedding different neural models in a VSR neural controller

Given the differences between the considered neural models, highlighted in Section 3, it logically follows that not all ANNs can be embedded in a VSR neural controller alike. In particular, a key role is played by the different ways in which information is stored and processed: MLPs and RNNs rely on real values at each neural simulation time step h , whereas SNNs encode information in spike trains and in their distribution over time. Therefore, we distinguish between the former and the latter models when used as VSR controllers.

5.1. MLPs and RNNs as VSR neural controllers

Using these ANNs as controllers is straightforward, as they process real values just like the VSR control system, which takes the sensor readings $\mathbf{r}^{(k)}$ as inputs and outputs the control values for the voxels $\mathbf{a}^{(k)}$. Therefore, it is sufficient to query those ANNs at each control time step k to compute the desired values. Hence, in this case, the neural simulation frequency coincides with the chosen control frequency f_c .

5.2. SNNs as VSR neural controllers

Conversely from canonical ANNs, SNNs do not process real values: at each neural time step h , there is a binary value, i.e., either there is a spike or not, which does not contain meaningful information itself. Instead, information is stored within the distribution of spikes over time, hence we need to convert real values to a spike distribution, i.e., to a spike train, and vice versa, in order to embed SNNs in a VSR controller. Here, we take inspiration from the most common conversion method, that is *rate coding* [52], where real values are mapped to frequencies, which are then used to generate spike trains [53], and vice versa. Moreover, it is interesting to note that rate coding contributes substantially to the homeostatic mechanism, which operates as a firing frequency regulator. We remark, though, that other types of coding exist, which naturally couple with other types of regulation mechanisms, as, e.g., temporal coding, which enhances the benefits of Spike-Timing-Dependent Plasticity (STDP) [54] for local learning [55].

As a consequence of the encoding choice, the neural simulation frequency in SNNs $f_h = \frac{1}{\Delta t_h}$ needs to be larger than the control frequency f_c ; i.e., the control interval has to be longer than the neural simulation interval. We set $f_h = 1$ kHz, corresponding to a time resolution of 1 ms, which is a commonly used value in the literature [31].

Depending on the value chosen for f_c we use two rate coding flavors. When $f_c = 4$ Hz, we use the standard version of rate coding. Given a sensor reading $r^{(k)}$ we convert it to a spike train as follows. We first compute $f^{(k)}$ by linearly scaling $r^{(k)}$ between f_{\min} and f_{\max} :

$$f^{(k)} = r^{(k)}(f_{\max} - f_{\min}) + f_{\min}. \quad (27)$$

Then we generate a spike train with spike frequency $f^{(k)}$, i.e., we emit a spike every $\lfloor \frac{f_h}{f^{(k)}} \rfloor$ neural time steps. Formally speaking, we assign each value $v^{(h)}$ in the spike train for $h = k \lfloor \frac{f_h}{f_c} \rfloor, \dots, (k + 1) \lfloor \frac{f_h}{f_c} \rfloor - 1$, i.e., for the k th control interval, as:

$$v^{(h)} = \begin{cases} 1 & \text{if } \exists n \in \mathbb{N} \text{ s.t. } h = h_0 + n \lfloor \frac{f_h}{f^{(k)}} \rfloor \\ 0 & \text{otherwise.} \end{cases} \quad (28)$$

where we set $h_0 = k \lfloor \frac{f_h}{f_c} \rfloor$ to the starting point of the control interval. Concerning the output conversion, we compute the average frequency of the output spike train in the control interval, the inverse of the average inter-spike timing, and we scale it considering the maximum frequency f_{\max} to compute $a^{(k)}$:

$$a^{(k)} = 2 \left(\frac{f_h}{f_{\max}} \sum_{h=k \lfloor \frac{f_h}{f_c} \rfloor}^{h=(k+1) \lfloor \frac{f_h}{f_c} \rfloor - 1} v^{(h)} \right) - 1. \quad (29)$$

Note that, since $a^{(k)}$ is defined in $[-1, 1]$, we multiply by 2 and subtract 1 to the rescaled value, which would otherwise be defined in $[0, 1]$. We set $f_{\min} = 5$ Hz and $f_{\max} = 50$ Hz for biological plausibility.

With the highest control frequency $f_c = 60$ Hz, we modify the standard rate coding to take into account the fact that $f_c \approx f_{\max}$. For the input conversion, we employ Eq. (28), but we set h_0 to the neural time step of the last spike occurred before this control interval. For the output conversion we modify Eq. (29) to consider the last n_w control intervals, i.e., from $k - n_w - 1$ to k , as in a moving average. Hence, the output conversion equation becomes

$$a^{(k)} = 2 \left(\frac{1}{n_w} \frac{f_h}{f_{\max}} \sum_{h=(k-n_w+1) \lfloor \frac{f_h}{f_c} \rfloor}^{h=(k+1) \lfloor \frac{f_h}{f_c} \rfloor - 1} v^{(h)} \right) - 1. \quad (30)$$

After preliminary experimentation, we set $n_w = 5$, as in [39].

We remark that the value-to-spikes and the spikes-to-value conversions are stateful. In particular, at $f_c = 60$ Hz, an unbounded number of previous control time steps may impact on the value-to-spikes conversion (since the last previous spike might have occurred in any previous control time step). In the spikes-to-value conversion, exactly n_w previous control time steps are considered.

6. Neuroevolution of VSR neural controllers

As described in the previous section, the controller of a VSR is a dynamical system defined by two parametric functions f_θ and g_θ . These functions determine the way in which the VSR reacts to different conditions, i.e., its behavior. Naturally, the goal of optimization is to find the most effective parameters θ to induce a behavior which leads the VSR to successfully accomplish a given task.

Since we are employing ANNs as controllers and evolutionary techniques for optimization, the process of optimizing the controller parameters θ can be referred to as *neuroevolution*. This means that we are searching the space of ANNs to find the most suitable ones for solving the task at hand. In this study, we keep the neural topology fixed throughout the entire optimization process, unlike other classical neuroevolutionary techniques, as e.g., NEAT [56,57], which also search the space of ANN topologies. Hence, here we consider the space \mathbb{R}^p of ANN numerical parameters θ as search space, and aim to find the optimum

therein. Although one could, in principle, include all possible ANN parameters in the search space, thus subjecting them to optimization, we limit ourselves to weights and biases (only weights for SNNs), keeping all other free parameters fixed to hand-set values (i.e., the activation function φ^p and SNN parameters Δt_h , λ_h , v_{rest} , λ_ψ , and ψ_{inc}).

Therefore, we reduce the search of an effective VSR neural controller to a numerical optimization problem, which we tackle with a simple variant of Evolution Strategies (ES) [58], described in Algorithm 1. We chose ES as they have not only been recognized as a scalable alternative to reinforcement learning for continuous control tasks [59], but they have also already proved successful for the neuroevolution of VSR controllers [7,23].

```

1 function evolve():
2   P ← ∅
3   foreach i ∈ {1, ..., n_pop} do
4     | P ← P ∪ {θ + U(-1, 1)p}
5   end
6   foreach g ∈ {1, ..., n_gen} do
7     | P_parents ← bestIndividuals(P, ⌊  $\frac{|P|}{4}$  ⌋)
8     | μ ← mean(P_parents)
9     | P' ← {bestIndividuals(P, 1)}
10    | while |P'| < n_pop do
11      | | P' ← P' ∪ {μ + N(0, σ)p}
12    | end
13    | P ← P'
14  end
15  return bestIndividuals(P, 1)
16 end

```

Algorithm 1: The simple ES used for neuroevolution.

The considered variant of ES operates on a population of p -dimensional numerical vectors θ . First, the population of size n_{pop} is initialized: each individual is randomly generated, sampling each component from a uniform distribution over the interval $[-1, 1]$. Then, the algorithm proceeds iteratively for n_{gen} iterations as follows. First, the fittest quarter of the population is selected as parents, and their element-wise mean μ is computed. Then, the fittest individual is promoted to the following generation. Last, each offspring is generated from μ by adding a random Gaussian noise $N(0, \sigma)$ to each of its components, until the new generation reaches the target size of n_{pop} . At the end of the iterative process, the fittest individual obtained is returned.

We remark that neuroevolution, i.e., the application of an evolutionary algorithm to optimize a neural network (here, just the parameters), is not the only approach for finding a good controller for a robotic agent performing a given task. In particular, in the last years, reinforcement learning proved to be a very practical alternative, also in scenarios where a model of the robot-environment systems is not known and where the input and output of the robots are numerical, vectors i.e., in continuous control tasks [60]. Interestingly, many of the most successful variants of reinforcement learning do, internally, exploit neural network, often being potentially rather large [61]: as such, those approaches are also forms of optimization in the space of (the parameters for) neural networks, as neuroevolution is. However, neuroevolution and (deep) reinforcement learning do differ in a key aspect: while for the former a single measure of quality (the fitness) is needed for capturing the degree of achievement of the task by the robot at the end of one entire attempt (the robot life, or the episode), for reinforcement learning a measure of achievement (the reward) should continuously be available to the robotic agent at every time step. In these terms, the reward

can be viewed as a form of supervision (possibly sparse, in practice), which is not needed by agents subjected to neuroevolution. Beyond this apparent difference, recent trends are making the boundary between reinforcement learning and (neuro)evolution more and more fuzzy [59], also for robotic control [62]. Indeed, reinforcement learning has been used also for VSRs, by Bhatia et al. [46]: the authors consider a few different tasks and describe the very specific reward function being used for each of them. Notably, in most cases that function exploits information (e.g., the absolute position of some parts of the robot) that is not perceived by the VSR through sensors, but is assumed to be known by an external entity which continuously feeds it to the robot, in the form of reward values: the practicality of such an approach should be verified in practice.

7. Experiments and discussion

We performed a thorough experimental evaluation to characterize the peculiarities of various neural models for controlling VSRs. In particular, we investigated along three quantitative axes, which are naturally intertwined: (a) effectiveness, (b) efficiency, and (c) generalization ability. Moreover, we also analyzed qualitatively the behavior of the evolved VSRs.

We applied neuroevolution to optimize the controller for 9 different simulated robots resulting from the combination of 3 morphologies and 3 sensory apparatuses. For each robot, we experimented with both high and low control frequencies, i.e., $f_c \in \{60 \text{ Hz}, 4 \text{ Hz}\}$, in combination with each one of the four proposed neural models.

We considered the task of locomotion, a common one in evolutionary robotics, for which we employed a flat terrain during evolution, and rougher terrains to test the generalization ability of the best individuals obtained.




Finally, we visually inspected and systematically analyzed the behavior of each robot, following the pipeline introduced by Pigozzi et al. [7].

We remark that, to properly put our results into context, we also included in our experimentation the current state-of-the-art controller for VSRs, that is the one involving the MLP as neural model and employing a control frequency of $f_c = 60 \text{ Hz}$.

7.1. Procedure and parameters

7.1.1. VSR morphologies and sensors

We experimented with three morphologies:

- *biped* , with 10 voxels enclosed in a 4×3 grid;
- *worm* , with 10 voxels enclosed in a 5×2 grid;
- *comb* , with 11 voxels enclosed in a 7×2 grid.

We equipped each morphology with three different sensory apparatuses:

- *touch*, with one touch sensor in each voxel in the bottom row of the grid;
- *sight*, with five proximity sensors in each voxel in the rightmost column of the grid, with α set to $-\frac{\pi}{4}$, $-\frac{\pi}{8}$, 0 , $\frac{\pi}{8}$ and $\frac{\pi}{4}$ rad and $d = 10 \text{ m}$ (for reference, the side of each voxel is 3 m long);
- *rich*, with the sensors of touch and sight, one x - and one y -velocity sensor in each voxel in the top row of the grid, and one area sensor in each voxel.

Fig. 2 shows the 9 VSR bodies resulting from the combination of the 3 morphologies and the 3 sensory apparatuses. Additionally, Table 1 summarizes, for each of the 9 VSR bodies, the number n of voxels (ranging from 10 to 11) and the number m of sensors (ranging from 2 to 39).

7.1.2. VSR neural controllers

Concerning the topology of the neural controllers, we employed the same number of hidden layers $l^* = 1$ for all the models and both control frequencies $f_c = 60 \text{ Hz}$ and $f_c = 4 \text{ Hz}$; we set the number of neurons in the single hidden layer to the number of inputs, i.e., $m_1 = m_0 = m$. This resulted in the number $|\theta|$ of evolvable parameters and the size $|s|$ of the ANN state showed in Table 1: $|\theta|$ ranges from 24 for case biped, touch, SNN (and SNN-H) to 3521 for the case comb, rich, RNN; $|s|$ ranges from 0, i.e., no state, for all the MLP-based cases to 267 for the case comb, rich, SNN-H. We recall that VSRs equipped with SNN and SNN-H neural controllers have some additional state used for the input and output conversion (see Section 5.2).

Fig. 3 summarizes the same data of Table 1 in the form of a scatter plot, where each marker corresponds to one case (morphology, sensory apparatus, neural model) and is colored according to the neural model. From this figure, the differences among neural models in terms of $|\theta|$ and $|s|$ are apparent. With the increasing complexity of the VSR body (i.e., with increasing m and n), RNNs are more expressive in terms of computation rather than of memory (i.e., the amount of parameters $|\theta|$ grows faster than the state size $|s|$), while SNNs are more expressive in terms of memory rather than of computation (i.e., $|s|$ grows faster than $|\theta|$).

Concerning the network-wise, non optimizable parameters of the neural models, we used tanh as the activation function φ^p (in MLP and RNN), $\Delta t_h = 1 \text{ ms}$, $\lambda_v = 0.01 \text{ s}^{-1}$, $v_{\text{rest}} = 0 \text{ mV}$ (in SNN and SNN-H), $\lambda_\psi = 0.01 \text{ s}^{-1}$, $\psi_{\text{inc}} = 0.2 \text{ mV}$ (in SNN-H), and $\vartheta_0 = 1 \text{ mV}$ (in SNN). Finally, in all the experiments, we set the initial state $\mathbf{s} = \mathbf{0}$, with the exception of the parts of the state corresponding to the membrane potentials in SNN and SNN-H, that we set, element-wise, to v_{rest} (which is 0 too).

7.1.3. Evolutionary optimization

We used the EA of Algorithm 1 with $n_{\text{pop}} = 36$, $\sigma = 0.35$, and $n_{\text{gen}} = 572$. We set n_{pop} based on the number of cores of the machines where we executed the experiments, in order to fully exploit the inherent capability of this EA to parallelize the fitness computation of the candidate solutions. We set n_{gen} based on n_{pop} in order to make each evolutionary run terminate after 20 000 fitness evaluations. We set σ based on our experience and on some exploratory experiments.

For each combination of morphology, sensory apparatus, neural model, and control frequency, we executed 10 independent evolutionary runs by varying the random seed. Overall, we executed $3 \cdot 3 \cdot 4 \cdot 2 \cdot 10 = 720$ evolutionary runs.

We optimized VSRs for the task of locomotion on a flat terrain. For each candidate solution θ , we initially placed a VSR equipped with a neural controller parameterized with θ on a flat terrain and let the simulation run for 30 s. We hence measured the average x -velocity v_x of the center of mass of the VSR during the last 25 s of the simulation and used it as the fitness of θ . We discarded the first 5 s of the simulation to let the neural controllers reach a steady gait regime; we also verified that the remaining 25 s were long enough to allow VSRs to exhibit an assessable behavior.

We limited ourselves to the locomotion task for it is a fundamental and challenging problem in evolutionary robotics, requiring the proper exploitation of the information coming from multiple sensors to effectively guide and coordinate the actuators. Moreover, locomotion is common to many real-world applications, where it can be declined in various nuances, e.g., climbing [46], or it can occur as sub-task, e.g., in exploring unknown environments through path planning [63,64] or carrying objects around [46].

For the experiments, we built a software framework based on JGEA [65] for the evolutionary optimization and on 2D-VSR-Sim [51] for the simulation of the VSRs. For the latter, we set

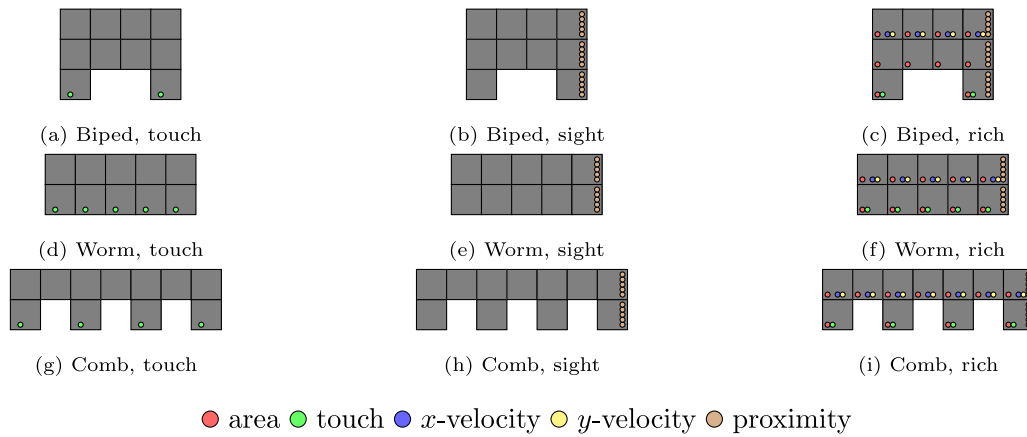


Fig. 2. VSR morphologies and sensory apparatuses.

Table 1

Number n of voxels and overall number m of sensors for the combinations of morphology and sensory apparatus. For each combination and each neural model, number $|\theta|$ of parameters and state size $|s|$.

Morphology	Sens. app.	m	n	Param. size $ \theta $				State size $ s $			
				MLP	RNN	SNN	SNN-H	MLP	RNN	SNN	SNN-H
Biped	Touch	2	10	36	40	24	24	0	2	14	42
	Sight	15	10	400	625	375	375	0	15	40	120
	Rich	35	10	1620	2845	1575	1575	0	35	80	240
Worm	Touch	5	10	90	115	75	75	0	5	20	60
	Sight	10	10	220	320	200	200	0	10	30	90
	Rich	35	10	1620	2845	1575	1575	0	35	80	240
Comb	Touch	4	11	75	91	60	60	0	4	19	57
	Sight	10	11	231	331	210	210	0	10	31	93
	Rich	39	11	2000	3521	1950	1950	0	39	89	267

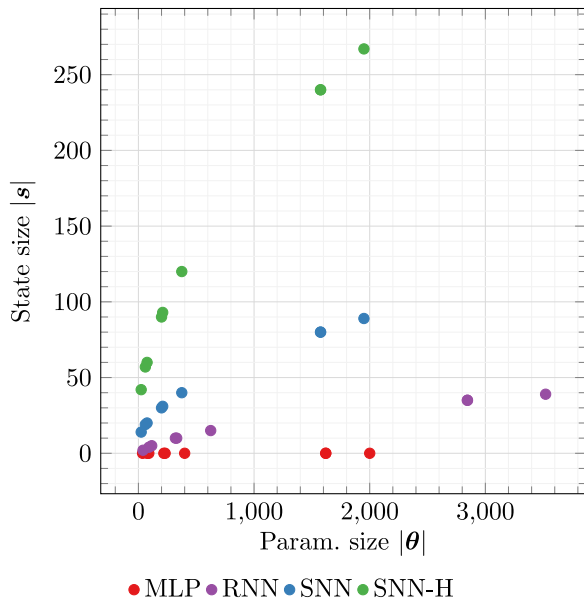


Fig. 3. Number $|\theta|$ of parameters and state size $|s|$ for each combination of morphology, sensory apparatus, and neural model. The four neural models differ in how $|s|$ and $|\theta|$ depend on the VSR body complexity (i.e., number n of voxels and overall number m of sensors).

each parameter to its default value, as we already witnessed no significant dependency of the results on them, provided they are kept fixed [12]. We made the software publicly available at <https://github.com/giorgia-nadizar/NeuralModelsVSR>. We performed the experiments on two workstations equipped with a

Intel[®] Xeon[®] CPU W-2295 with 36 cores running at 3.00 GHz to 4.80 GHz and with 64 GB RAM. On this hardware, one simulation of 30 s (simulated time) lasted approximately 0.9 s (wall time).

7.2. Effectiveness of evolved neural controllers

We consider as search effectiveness of the neuroevolution the effectiveness of the evolved neural controllers, namely, the fitness v_x^* of the best individual at the end of the evolutionary run.

Fig. 4 shows v_x^* for the 72 different cases in the form of a matrix of box-plots. In particular, for each combination of morphology (row of plot), sensory apparatus and control frequency f_c (column of plots), and neural model (box color), the figure considers the 10 values for v_x^* obtained from the 10 runs with different random seeds. We recall that for v_x^* , the greater, the better.

For what concerns the neural models, Fig. 4 shows that RNN gives the best results, i.e., the fastest robots: the median v_x^* is sharply larger than the one obtained with the other neural models in almost all the cases. To further validate this finding, we performed a number of statistical significance tests, as follows: for each combination of morphology, sensory apparatus, control frequency, and each pair of neural models, we carried out a one-sided Mann Whitney U rank test (after having verified the adequate hypotheses) with the null hypothesis that the distribution of v_x^* for the first neural model is stochastically lower or equal than the distribution of v_x^* for the second neural model. Then, with a significance level of $\alpha = 0.05$ (with Bonferroni correction with $n = 12$), we counted, for each pair of neural models, the number of combinations for which the null hypothesis is rejected. Table 2 shows the outcome of this procedure. The table confirms the findings suggested by Fig. 4: RNN is significantly better than the other models in the vast majority of the 18 cases and is not significantly worse than any other model.

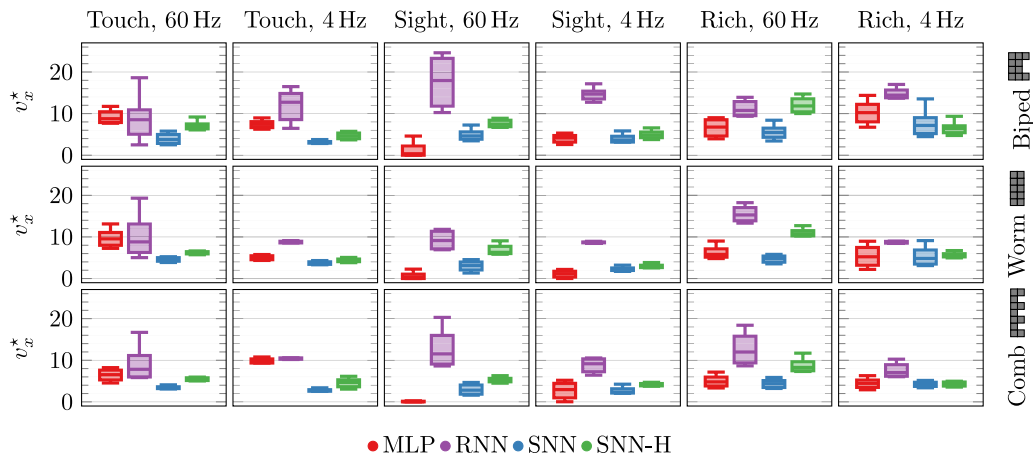


Fig. 4. Fitness v_x^* of the best individual at the end of the evolution.

Table 2

Number of cases (i.e., combinations of morphology, sensory apparatus, and control frequency) for which the neural model on the row is statistically significantly better (in terms of v_x^* , see text) than the neural model on the column. The last column shows the overall number of pairwise comparison for which a neural model is statistically significantly better than one other neural model.

	MLP	RNN	SNN	SNN-H	Tot.
MLP	–	0	6	5	11
RNN	12	–	17	14	43
SNN	4	0	–	0	4
SNN-H	7	0	13	–	20

Besides highlighting the sharp differences in median v_x^* among neural models, Fig. 4 also shows that RNN tends to have the largest variability in v_x^* (vertical extent of the boxes). Nevertheless, this neural model often outperforms the other models even in the worst evolutionary run.

If we restrict the comparison to the other three models, the experiments summarized in Fig. 4 suggest that there is no clear winner. In particular, MLP and SNN-H outperform each other in a similar number of cases: yet, they appear to perform best in different conditions (more on this later). Finally, both Fig. 4 and Table 2 show that homeostasis is beneficial: for the majority of cases, SNN-H outperforms SNN, while the opposite never happens.

A second set of considerations can be made by observing Fig. 4 in terms of the sensory apparatus. Sight looks the case for which the differences among neural models are the most consistent when the morphology and control frequency vary. In particular, neural models with larger states (i.e., $|s|$) appear to perform better with this sensory apparatus. This finding seems to agree with some previous studies [66] on simulated embodied agents that suggested that long-term planning is more important when coupled with fine-grained sensing, like long-range sight. However, we believe that in our scenario, i.e., locomotion on a flat terrain, the long-term planning ability of an agent does not play a significant role. Instead, we explain our finding in terms of the kind of information the sight sensory apparatus provides: for most time, most of the proximity sensors just return 1, since there are no objects to be seen. Only when the robot bends in such a way that the sensors “see” the terrain, they return a different value. Hence, the information coming from sight is “sparse” in time: in order to exploit this information for forming an effective gait, the VSR has to store it for a while; for this reason, a large memory is beneficial. Our interpretation is corroborated by two other observations. First, with sight, MLP is always worse at $f_c =$

60 Hz than $f_c = 4$ Hz: the fact that with the lower frequency the control values are kept constant for a longer time somehow mitigates the inability of MLP to store information, i.e., its lack of state. Second, with touch, MLP is never worse than SNN or SNN-H: here, the sensory information induces a purely reactive behavior (i.e., expand some voxels as soon as they touch the ground) which is much more effective and hence the memory is not particularly useful.

Finally, for what concerns the morphologies, Fig. 4 shows that the biped is the most effective morphology in locomotion, as it has been observed in some previous studies [5,24]. However, neuroevolution is able to find controllers that generate effective gaits in almost all the cases (with the exceptions of MLP with sight and $f_c = 60$ Hz for the worm and the comb).

7.3. Efficiency of neuroevolution

We consider as search efficiency of the neuroevolution the effort taken to find an effective solution. Since a definition of effective solution based on the absolute value of the fitness would hardly fit all the considered cases, we consider as effective a solution whose fitness is $\geq 80\%$ of the fitness v_x^* of the best individual at the end of the corresponding run. This way, we define the efficiency as the number g_{80} of generations the EA takes to find the first effective solution.

Fig. 5 shows how the fitness v_x of the best individual varies during the evolution for the 72 cases. For each combination of morphology, sensory apparatus, control frequency, and neural model, the figure shows the median v_x across the 10 runs vs. the number of generations, i.e., iterations of the EA of Algorithm 1. Based on the data backing Fig. 5, which confirms the findings drawn in the previous section, we computed g_{80} for each run. Fig. 6 shows g_{80} for the 72 cases in the form of a matrix of box-plots, organized in the same way as Fig. 4.

By observing Fig. 6, it can be seen that the differences among neural models in terms of efficiency of the search are less apparent than the ones in terms of effectiveness. However, RNN again appears to be the preferable neural model, since it results in a more efficient search, i.e., lower g_{80} , in many cases. This is confirmed also by the statistical significance analysis we carried out in the same way we did for v_x^* , whose outcome is summarized in Table 3. We highlight the fact that RNN has a lower g_{80} in many cases, despite being also the model with the largest effectiveness v_x^* : that is, it allows to find the fastest VSR in the shortest time. This can be appreciated by observing Fig. 5, e.g., for biped, rich, 4 Hz. There are, however, cases where the neuroevolution takes

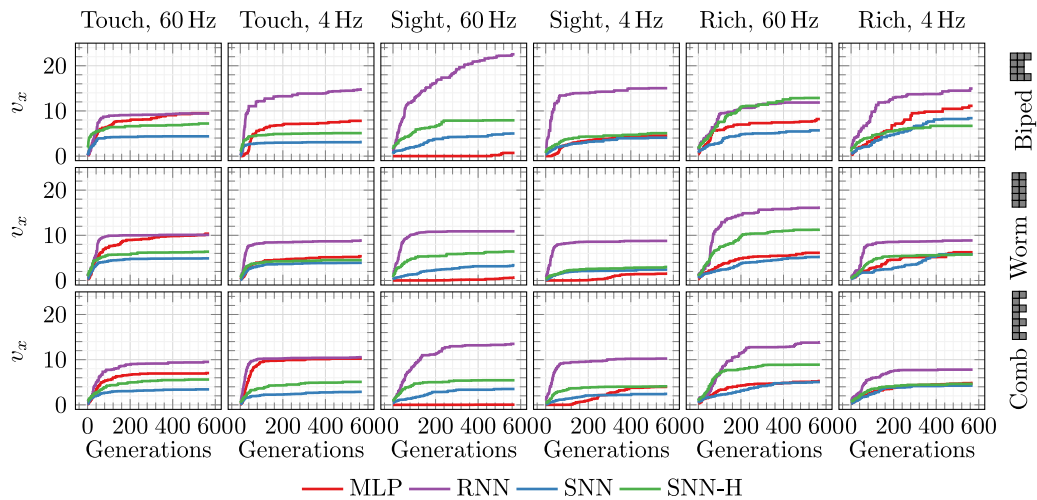


Fig. 5. Fitness v_x during the evolution.

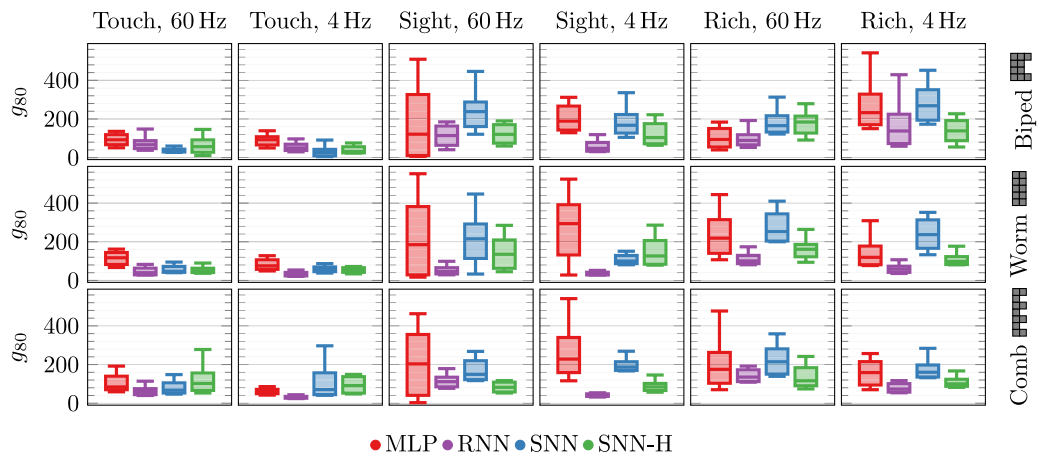


Fig. 6. Efficiency measured with g_{80} , defined as the number of generations needed to reach a solution with fitness $v_x \geq 0.8v_x^*$.

Table 3

Number of cases (i.e., combinations of morphology, sensory apparatus, and control frequency) for which the search with the neural model on the row is statistically significantly more efficient (in terms of g_{80} , see text) than the search with the neural model on the column. The last column shows the overall number of pairwise comparison for which a neural model is statistically significantly better than one other neural model.

	MLP	RNN	SNN	SNN-H	Tot.
MLP	–	0	0	0	0
RNN	7	–	9	5	21
SNN	3	1	–	0	4
SNN-H	3	0	4	–	7

longer to find an effective solution with RNN, than with other models, e.g., for biped, sight, 60 Hz.

For what concerns the other models, Table 3 and Fig. 6 show that (a) MLP never results in a faster search than the other models and (b) homeostasis is beneficial also for efficiency: SNN-H is never worse than SNN and outperforms it in a few cases.

Beyond neural models, Fig. 6 suggests that – not surprisingly – the number $|\theta|$ of parameters does impact the search efficiency; in turn, $|\theta|$ depends mainly on the number m of sensors in the VSR, as shown earlier in Table 1. For touch, that corresponds to $|\theta| \in [24, 115]$, the search is in general very fast: 100 generations are often enough to find an effective solution. For the sight and rich sensory apparatuses, for which $|\theta| \in [210, 625]$ and

$|\theta| \in [1575, 3521]$ respectively, the g_{80} is not as different as one could expect, given the rather large difference in $|\theta|$. We hypothesize that this is motivated by the fact that searching for a good controller is harder with sight only: from another point of view, while the search space for sight is smaller than the one for rich, the fitness landscape for the former might be harder to be explored. We leave the investigation on how the sensory apparatus impacts on the fitness landscape to future work.

7.4. Generalization ability

We define as generalization ability the ability of an evolved neural controller to retain its effectiveness when operating in conditions different from the ones it has been evolved in. For the task of locomotion, we consider the x -velocity as effectiveness and the shape of the terrain as conditions.

In particular, for measuring the generalization ability of the four neural models coupled with the different morphologies, sensory apparatuses, and control frequencies, we took the best VSR obtained at the end of each evolutionary run (i.e., the one giving v_x^* for that run) and measured its x -velocity on a set of 16 *unseen* terrains, different from the flat one used for measuring the fitness. We considered uphill and downhill terrains with an even surface, and terrains with an uneven surface, including some with small steps and some with small hills. We collected the 16 v_x values from the simulations (computed discarding the initial 5s,

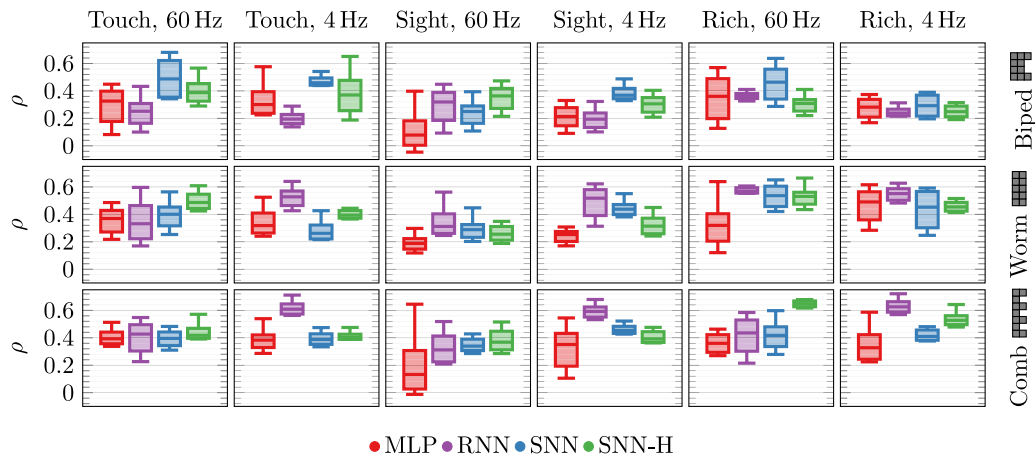


Fig. 7. Generalization ability ρ of the evolved neural controllers.

Table 4

Number of cases (i.e., combinations of morphology, sensory apparatus, and control frequency) for which the neural model on the row is statistically significantly better (in terms of ρ , see text) than the neural model on the column. The last column shows the overall number of pairwise comparison for which a neural model is statistically significantly better than one other neural model.

	MLP	RNN	SNN	SNN-H	Tot.
MLP	–	0	0	0	0
RNN	7	–	4	4	15
SNN	2	3	–	2	7
SNN-H	4	4	2	–	10

as for the evolution) and averaged them, obtaining one \bar{v}_x for each VSR. Finally, we computed the ratio $\rho = \frac{\bar{v}_x}{v_x^*}$ between the average x -velocity on the unseen terrains and the one obtained on the flat terrain: ρ represents the degree to which the effectiveness in locomotion is retained in different conditions, hence, it represents the generalization ability of a neural controller.

Fig. 7 shows ρ for the 72 different cases in the form of a matrix of box-plots, organized in the same way as Figs. 4 and 6. We recall that also for ρ , like for v_x^* , the greater, the better.

By comparing, at a glance, Fig. 7 against Fig. 4, it is apparent that the differences among neural models are smaller for ρ than for v_x^* . This is further confirmed by Table 4, that summarizes the outcome of a statistical significance analysis we carried out in the same way we did for v_x^* in Tables 2 and 3.

On average, the x -velocity achieved by best VSRs on the unseen terrains is 40% of the one achieved on the flat terrain. In terms of generalization ability, RNN does not outperform the other neural models as neatly as for v_x^* : while, in practice, VSRs equipped with RNN are still the fastest also on unseen terrains, ρ is comparatively smaller. We believe this can be explained by the fact that the gaits obtained with RNN were highly optimized for the flat terrain and hence the corresponding VSRs struggled more in different conditions.

On the other hand, MLP still appears to be the worst neural model. Table 4 shows that it is the being outperformed by the other models in the largest number of cases. Moreover, despite having obtained low v_x^* values when coupled with sight at $f_c = 60$ Hz, in those cases it is still the worst model also in terms of ρ . Since MLP is the only model without a state, we interpret this finding as a clue that memory is beneficial to generalization ability.

Finally, the differences between SNN and SNN-H are negligible in terms of ρ , according to Fig. 7 and Table 4. That is, homeostasis brings a clear advantage in terms of effectiveness without corresponding to any significant drop in generalization ability.

7.5. Behavioral analysis

Besides the quantitative analysis devoted to assessing the performance of the neural models, it is relevant to characterize, on a more qualitative level, the behaviors each of them induces when coupled with different VSR morphologies and sensory apparatuses. Given the large amount of combinations considered, though, it would be impossible to visually inspect all the evolved VSRs, yet, with an overview, we noticed some core differences worth systematically analyzing.

To this extent, we relied on some behavioral features automatically extracted from the gait of the VSR in a simulation, based on the position and orientation of its voxels over time [7]. More in detail, we proceeded as follows:

- (i) for each i th voxel of the VSR, we considered the signals $x_i^{(k)}$, $y_i^{(k)}$, $\beta_i^{(k)}$ of its center x - and y -coordinates and of its rotation β —for the latter, we consider the voxel initial (i.e., at $k = 0$) rotation as reference;
- (ii) for each time step and each of the three signals, we computed the average across all the voxels, hence obtaining three VSR-wise signals $x^{(k)}$, $y^{(k)}$, $\beta^{(k)}$;
- (iii) we computed the first differences of the three signals, obtaining $\Delta x^{(k)}$, $\Delta y^{(k)}$, $\Delta \beta^{(k)}$;
- (iv) for each one of the three signals, we calculated the Fast Fourier Transform (FFT), from which we took the magnitude, filtered out frequency components not in the range [0 Hz, 4 Hz], and re-sampled the remaining components to have 8 of them for each signal.

From the concatenation of these $3 \cdot 8$ components we obtained the final *behavior vector*, i.e., the feature vector describing the behavior of the VSR in a simulation.

We applied this feature extraction procedure to the gait of the best VSR obtained from each evolutionary run. As during evolution, we conducted a simulation of 30 s on a flat terrain, but we considered only the last 25 s for obtaining the behavior vector of the VSR, assuming the discarded 5 s are enough to reach a steady gait.

In order to enable the comparative visualization of the behavior vectors, we performed a dimensionality reduction using the principal component analysis (PCA). In particular, we grouped the behavior vectors by VSR morphology and, after standardizing them, we relied on the PCA to perform a dimensionality reduction from 24 to 2. For the biped and the comb morphologies, the reduction captured around 60% of the total variance, whereas for the worm, it captured around 50%. For each of the morphologies, we visualized the first two components obtained in a scatter plot,

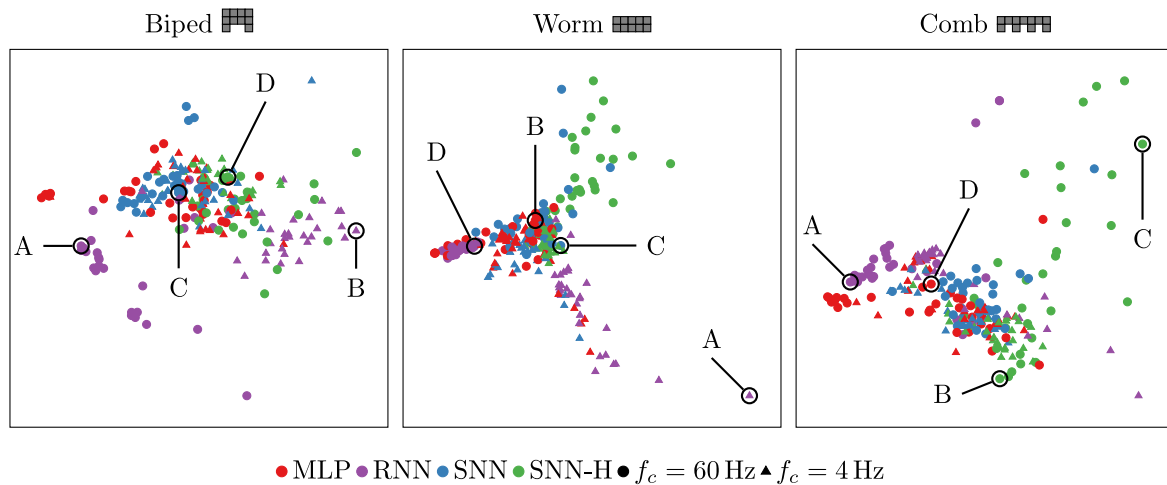


Fig. 8. Behaviors PCA. Note that PCA was performed independently for each shape, i.e., the principal components are not the same in the plots. The letters in each plot refer to the individuals we sampled for visual inspection: the videos of the simulations are available at <https://giorgia-nadizar.github.io/NeuralModelsVSR/>. For the biped, the four behaviors are shown in the form of time-lapses in Fig. 1.

distinguishing neural models with different colors, and control frequencies with different markers. We report the scatter plots for the three morphologies in Fig. 8.

For each of the plots in the figure we can notice that the points appear to be mostly concentrated in one area, with some scattering around. This can be considered as an indicator that each morphology is associated to one main effective behavior or to a range of similar effective behaviors, yet there is some room for diversity. Focusing on the first plot, corresponding to the biped morphology, we see a great overlap between MLPs and SNNs at the center of the plane. Moving away from the center, we see RNNs with $f_c = 60$ Hz neatly separated on the left side, whereas RNNs with low control frequency appear akin to SNN-H, slightly to the right. Concerning the worm morphology, we see a higher density at the center, but with multiple tails: SNN-H with $f_c = 60$ Hz with some SNNs on the top right, mostly RNNs with $f_c = 4$ Hz at the bottom, and a mixture of MLPs and RNNs on the left. Last, for the comb morphology, we have the central area mostly occupied by MLPs and SNNs, MLPs and RNNs with $f_c = 60$ Hz on the left, and a scatter of SNN-H on the right. Focusing on the global picture, i.e., considering all the morphologies, we can note that RNNs and SNN-H usually appear to move away from the center, i.e., they tend to induce more diverse behaviors, especially for the worm and the comb. Moreover, we see that for these models, the control frequency plays a more important role in determining the VSR behavior than for MLPs and SNNs. This might be a hint that the larger the state, the more important the control frequency becomes in guiding the behavior. Such finding is more difficult to notice from bare performance results, as, in general, evolution tends to find a way to achieve good results, eventually making up for less effective behaviors with a finer tuning of parameters.

Finally, we manually selected 4 evolved VSRs for each morphology that corresponded to distant points in the PCA plots of Fig. 8—we marked those points with capital letters in the figure. We analyzed the 12 behaviors and verified that they are indeed qualitatively different. We made the videos of the corresponding simulations publicly available at <https://giorgia-nadizar.github.io/NeuralModelsVSR/>. One of the VSR for each morphologies exhibited a vibrating behavior, namely A for the biped, D for the worm, A for the comb. Interestingly, in all the three cases, the VSRs were equipped with an RNN operating at $f_c = 60$ Hz: indeed, due to their nature, in which hidden neurons are fed with their activation value at the previous step, RNNs are particularly prone

to generate this kind of behavior. However, while in our experiments vibration is effective for locomotion, real VSRs (maybe fabricated with the techniques described in [67,68]) might be unable to perform an effective locomotion just by vibrating. From this point of view, RNNs appear to be more prone to the reality gap problem [69,70] than SNNs and MLPs, when operating at $f_c = 60$ Hz.

8. Discussion and limitations

Through our experimental analysis we investigated how different neural models are intertwined with the VSRs performance and behavior for the locomotion task. Namely, we observed that in almost all cases it was possible to achieve effective gaits thanks to neuroevolution, regardless of the VSR morphology, sensory apparatus, or neural model. However, it was clear that some neural models couple with the VSR dynamics more successfully than others. Namely, those with state, i.e., RNN and SNN (with and without homeostasis), proved to be better at exploiting the complex dynamics deriving from soft materials, thus achieving greater effectiveness and more diverse behaviors. Yet, in the cases where an instinctive behavior sufficed, e.g., if the sensory information could be immediately exploited, we noticed that even more simple models like MLPs could carry out the locomotion task well.

As this study was conducted in simulation only, one noteworthy limitation derives from not testing the controllers in the real-world. In fact, there have been various attempts at building VSRs, starting from that of Hiller and Lipson [4], which initially proposed this type of robots. After this germinal work, several other groups have pursued physical implementations of modular soft robots, as [71,67,68], the most ambitious of them even relying on living matter [72]. However, to this date, none of the existing physical VSRs can be finely controlled with closed-loop controllers as in this study, making it still unfeasible to bring this experimentation to the real-world. As a matter of fact, it remains unclear whether the observations made in simulation could still hold for real robots, given the wide range of (sometimes unpredictable) dynamics of actual soft materials. Nevertheless, we deem this study as a useful starting point for practitioners, who can benefit from the high level observations drawn, as, e.g., those regarding the importance of the controller memory in some circumstances.

Another potential limitation of this study lies in the consideration of the locomotion task only. Namely, it is not clear if the

findings deriving from this experimental evaluation can be ported to other scenarios. However, it is worth to mention that most of the relevant tasks for modular soft robots, e.g., VSRs, consist of either variants of locomotion or involve locomotion as sub-task [46]. Hence, our results can be a source of knowledge to build upon when considering locomotion-based tasks. Moreover, some of the made observations remain valid also for completely different tasks, as, e.g., the possibility of exploiting instinctive behaviors when memory is not available.

9. Concluding remarks

Choosing the most effective neural network models for robotic control is an open question. This is particularly challenging for modular and soft robots, due to their intricate dynamics emerging from the interactions of the components in response to their morphological computation, the available sensory apparatus, and the responses from the environment. In this work, we have carried out a systematic experimental campaign involving simulated modular robots to shed the light onto the benefits of different neural network models, including networks without and with state. Our main findings show that networks with recurrent connections are generally beneficial in terms of effectiveness, specifically for the task of locomotion of voxel-based soft robots. However, spiking neural networks provide often better ability to generalize to changes in the environment. This is particularly true when a mechanism of homeostatic plasticity is introduced, which provides an opportunity of fast adaptation during the lifetime of the robotic agent.

As future work, we plan to address some of the limitations of this study, with the goal of synthesizing more general observations on the potential of different neural models for robot control. Namely, we aim at experimenting in different scenarios, taking inspiration from the tasks of [46] and extending further to other tasks which also require more high level capabilities as, e.g., planning ahead. Moreover, we plan to extend our study to other instances of simulated modular robots [73,74,46], both soft and not, to assess the consistency of outcomes across different simulators and robots. Last, it would be noteworthy to consider neural models with unsupervised learning paradigms in combination with neuroevolution, as Hebbian Learning for MLPs [24] or STDP [54] for SNNs.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

- [1] J. Schulman, B. Zoph, C. Kim, J. Hilton, J. Menick, J. Weng, J. Uribe, L. Fedus, L. Metz, M. Pokorny, et al., ChatGPT: Optimizing language models for dialogue, 2022, OpenAI. com <https://openai.com/blog/chatgpt>.
- [2] M.F.A.R.D.T. (FAIR), A. Bakhtin, N. Brown, E. Dinan, G. Farina, C. Flaherty, D. Fried, A. Goff, J. Gray, H. Hu, et al., Human-level play in the game of diplomacy by combining language models with strategic reasoning, *Science* 378 (2022) 1067–1074.
- [3] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, A. Zeng, Code as policies: Language model programs for embodied control, 2022, arXiv Preprint [arXiv:2209.07753](https://arxiv.org/abs/2209.07753).
- [4] J. Hiller, H. Lipson, Automatic design and manufacture of soft robots, *IEEE Trans. Robot.* 28 (2012) 457–466.
- [5] J. Talamini, E. Medvet, S. Nichele, Criticality-driven evolution of adaptable morphologies of voxel-based soft-robots, *Front. Robot. AI* 8 (2021) 172.
- [6] V.C. Müller, M. Hoffmann, What is morphological computation? On how the body contributes to cognition and control, *Artif. Life* 23 (2017) 1–24.
- [7] F. Pigozzi, E. Medvet, A. Bartoli, M. Rochelli, Factors impacting diversity and effectiveness of evolved modular robots, *ACM Trans. Evol. Learn.* 3 (2023) 1–33.
- [8] S.M. Prabhu, D.P. Garg, Artificial neural network based robot control: An overview, *J. Intell. Robot. Syst.* 15 (1996) 333–365.
- [9] G.A. Bekey, K.Y. Goldberg, *Neural Networks in Robotics*, vol. 202, Springer Science & Business Media, 2012.
- [10] R. Pfeifer, J. Bongard, *How the Body Shapes the Way We Think: A New View of Intelligence*, MIT Press, 2006.
- [11] C. Della Santina, R.K. Katzschmann, A. Biechi, D. Rus, Dynamic control of soft robots interacting with the environment, in: 2018 IEEE International Conference on Soft Robotics, RoboSoft, IEEE, 2018, pp. 46–53.
- [12] E. Medvet, G. Nadizar, F. Pigozzi, On the impact of body material properties on neuroevolution for embodied agents: The case of voxel-based soft robots, in: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022, pp. 2122–2130.
- [13] K. Walker, H. Hauser, S. Risi, Growing simulated robots with environmental feedback: An eco-evo-devo approach, in: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2021, pp. 113–114.
- [14] G. Nadizar, E. Medvet, K. Miras, On the schedule for morphological development of evolved modular soft robots, in: *European Conference on Genetic Programming, Part of EvoStar*, Springer, 2022, pp. 146–161.
- [15] K. Horibe, K. Walker, S. Risi, Regenerating soft robots through neural cellular automata, in: *Genetic Programming: 24th European Conference, EuroGP 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24*, Springer, 2021, pp. 36–50.
- [16] A.L. Freire, G.A. Barreto, M. Veloso, A.T. Varela, Short-term memory mechanisms in neural network learning of robot navigation tasks: A case study, in: 2009 6th Latin American Robotics Symposium, LARS 2009, IEEE, 2009, pp. 1–6.
- [17] E. Najarro, S. Risi, Meta-learning through hebbian plasticity in random networks, *Adv. Neural Inf. Process. Syst.* 33 (2020) 20719–20731.
- [18] J.W. Pedersen, S. Risi, Evolving and merging hebbian learning rules: Increasing generalization by decreasing the number of rules, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, 2021, pp. 892–900.
- [19] A. Eiben, E. Hart, If it evolves it needs to learn, in: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, 2020, pp. 1383–1384.
- [20] L. Jin, S. Li, J. Yu, J. He, Robot manipulator control using neural networks: A survey, *Neurocomputing* 285 (2018) 23–34.
- [21] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.* 5 (1943) 115–133.
- [22] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychol. Rev.* 65 (1958) 386.
- [23] G. Nadizar, E. Medvet, H.H. Ramstad, S. Nichele, F.A. Pellegrino, M. Züllich, Merging pruning and neuroevolution: Towards robust and efficient controllers for modular soft robots, *Knowl. Eng. Rev.* 37 (2022).
- [24] A. Ferigo, G. Iacca, E. Medvet, F. Pigozzi, Evolving Hebbian learning rules in voxel-based soft robots, *IEEE Trans. Cogn. Dev. Syst.* (2022).
- [25] T.G. Thuruthel, B. Shih, C. Laschi, M.T. Tolley, Soft robot perception using embedded soft sensors and recurrent neural networks, *Sci. Robot.* 4 (2019) eaav1488.
- [26] M. Lechner, R. Hasani, M. Zimmer, T.A. Henzinger, R. Grosu, Designing worm-inspired neural networks for interpretable robotic control, in: 2019 International Conference on Robotics and Automation, ICRA, IEEE, 2019, pp. 87–94.
- [27] K. Akinci, A. Philippides, Evolving recurrent neural network controllers by incremental fitness shaping, in: *Artificial Life Conference Proceedings*, 2019, pp. 416–423.
- [28] X. Zou, E. Scott, A. Johnson, K. Chen, D. Nitz, K. De Jong, J. Krichmar, Neuroevolution of a recurrent neural network for spatial and working memory in a simulated robotic environment, in: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2021, pp. 289–290.
- [29] J. Nielsen, H.H. Lund, Spiking neural building block robot with Hebbian learning, IROS 2003(Cat. No. 03CH37453), in: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, IEEE, 2003, pp. 1363–1369.
- [30] W. Gerstner, W.M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, 2002.
- [31] E.M. Izhikevich, Which model to use for cortical spiking neurons? *IEEE Trans. Neural Netw.* 15 (2004) 1063–1070.
- [32] G.G. Turrigiano, S.B. Nelson, Homeostatic plasticity in the developing nervous system, *Nature Rev. Neurosci.* 5 (2004) 97–107.
- [33] A. Spaeth, M. Tebyani, D. Haussler, M. Teodorescu, Neuromorphic closed-loop control of a flexible modular robot by a simulated spiking central pattern generator, in: 2020 3rd IEEE International Conference on Soft Robotics, RoboSoft, IEEE, 2020, pp. 46–51.

- [34] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, A.C. Knoll, A survey of robotics control based on learning-inspired spiking neural networks, *Front. Neurobotics* 12 (2018) 35.
- [35] A. Spaeth, M. Tebyani, D. Haussler, M. Teodorescu, Spiking neural state machine for gait frequency entrainment in a flexible modular robot, *PLoS One* 15 (2020) e0240267.
- [36] A.J. Ijspeert, Central pattern generators for locomotion control in animals and robots: A review, *Neural Netw.* 21 (2008) 642–653.
- [37] B. Strohmer, P. Manoonpong, L.B. Larsen, Flexible spiking cpgs for online manipulation during hexapod walking, *Front. Neurobotics* 14 (2020) 41.
- [38] T. Klärner, E.P. Zehr, Sherlock holmes and the curious case of the human locomotor central pattern generator, *J. Neurophysiol.* 120 (2018) 53–77.
- [39] G. Nadizar, E. Medvet, S. Nichele, S. Pontes-Filho, Collective control of modular soft robots via embodied spiking neural cellular automata, 2022, arXiv Preprint arXiv:2204.02099.
- [40] J. Bono, S. Zannone, V. Pedrosa, C. Clopath, Learning predictive cognitive maps with spiking neurons during behavior and replays, *Elife* 12 (2023) e80671.
- [41] S.A. Lobov, A.I. Zharinov, V.A. Makarov, V.B. Kazantsev, Spatial memory in a spiking neural network with robot embodiment, *Sensors* 21 (2021) 2678.
- [42] M.N. Zennir, M. Benmohammed, R. Boudjadja, Spike-time dependant plasticity in a spiking neural network for robot path planning, in: *AAAI Workshops*, 2015, pp. 2–13.
- [43] N.A. Bomberger, A.M. Waxman, B.J. Rhodes, N.A. Sheldon, A new approach to higher-level information fusion using associative learning in semantic networks of spiking neurons, *Inf. Fusion* 8 (2007) 227–251.
- [44] S.A. Lobov, A.N. Mikhaylov, M. Shamshin, V.A. Makarov, V.B. Kazantsev, Spatial properties of STDP in a self-learning spiking neural network enable controlling a mobile robot, *Front. Neurosci.* 14 (2020) 88.
- [45] J. Li, D. Li, R. Jiang, R. Xiao, H. Tang, K.C. Tan, Vision-action semantic associative learning based on spiking neural networks for cognitive robot, *IEEE Comput. Intell. Mag.* 17 (2022) 27–38.
- [46] J. Bhatia, H. Jackson, Y. Tian, J. Xu, W. Matusik, Evolution gym: A large-scale benchmark for evolving soft robots, *Adv. Neural Inf. Process. Syst.* 34 (2021) 2201–2214.
- [47] S.M.J. Jalali, S. Ahmadian, A. Khosravi, S. Mirjalili, M.R. Mahmoudi, S. Nahavandi, Neuroevolution-based autonomous robot navigation: A comparative study, *Cogn. Syst. Res.* 62 (2020) 35–43.
- [48] T. Mildenerger, Stephen marsland: Machine learning. An algorithmic perspective, *Statist. Papers* 55 (2014) 575.
- [49] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural Netw.* 2 (1989) 359–366.
- [50] Y. Lu, W. Wang, L. Xue, A hybrid CNN-LSTM architecture for path planning of mobile robots in unknow environments, in: *2020 Chinese Control and Decision Conference, CCDC, IEEE*, 2020, pp. 4775–4779.
- [51] E. Medvet, A. Bartoli, A. De Lorenzo, S. Seriani, 2D-VSR-sim: A simulation tool for the optimization of 2-D voxel-based soft robots, *SoftwareX* 12 (2020) 100573.
- [52] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, E. Beigne, Spiking neural networks hardware implementations and challenges: A survey, *ACM J. Emerg. Technol. Comput. Syst.* 15 (2019) 1–35.
- [53] X. Wang, Z.-G. Hou, A. Zou, M. Tan, L. Cheng, A behavior controller based on spiking neural networks for mobile robots, *Neurocomputing* 71 (2008) 655–666.
- [54] Y. Li, Y. Zhong, J. Zhang, L. Xu, Q. Wang, H. Sun, H. Tong, X. Cheng, X. Miao, Activity-dependent synaptic plasticity of a chalcogenide electronic synapse for neuromorphic systems, *Sci. Rep.* 4 (2014) 1–7.
- [55] W. Guo, M.E. Fouda, A.M. Eltawil, K.N. Salama, Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems, *Front. Neurosci.* 15 (2021) 638474.
- [56] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, *Evol. Comput.* 10 (2002) 99–127.
- [57] K.O. Stanley, R. Miikkulainen, Efficient reinforcement learning through evolving neural network topologies, in: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, 2002, pp. 569–577.
- [58] H.-G. Beyer, H.-P. Schwefel, Evolution strategies—a comprehensive introduction, *Nat. Comput.* 1 (2002) 3–52.
- [59] T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever, Evolution strategies as a scalable alternative to reinforcement learning, 2017, arXiv Preprint arXiv:1703.03864.
- [60] B. Recht, A tour of reinforcement learning: The view from continuous control, *Annu. Rev. Control Robot. Auton. Syst.* 2 (2019) 253–279.
- [61] Y. Li, Deep reinforcement learning: An overview, 2017, arXiv Preprint arXiv:1701.07274.
- [62] F. Stulp, O. Sigaud, Robot skill learning: From reinforcement learning to evolution strategies, *Paladyn J. Behav. Robot.* 4 (2013) 49–61.
- [63] U. Orozco-Rosas, O. Montiel, R. Sepúlveda, Mobile robot path planning using membrane evolutionary artificial potential field, *Appl. Soft Comput.* 77 (2019) 236–251.
- [64] U. Orozco-Rosas, K. Picos, J.J. Pantrigo, A.S. Montemayor, A. Cuesta-Infante, Mobile robot path planning using a QAPF learning algorithm for known and unknown environments, *IEEE Access* 10 (2022) 84648–84663.
- [65] E. Medvet, G. Nadizar, L. Manzoni, JGEA: A modular java framework for experimenting with evolutionary computation, in: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022, pp. 2009–2018.
- [66] S. Pratt, L. Weihs, A. Farhadi, The introspective agent: Interdependence of strategy, physiology, and sensing for embodied agents, 2022, arXiv Preprint arXiv:2201.00411.
- [67] X. Sui, H. Cai, D. Bie, Y. Zhang, J. Zhao, Y. Zhu, Automatic generation of locomotion patterns for soft modular reconfigurable robots, *Appl. Sci.* 10 (2020) 294.
- [68] J. Legrand, S. Terryn, E. Roels, B. Vanderborght, Reconfigurable, multi-material, voxel-based soft robots, *IEEE Robot. Autom. Lett.* (2023).
- [69] E. Salvato, G. Fenu, E. Medvet, F.A. Pellegrino, Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning, *IEEE Access* (2021).
- [70] F. van Diggelen, E. Ferrante, N. Harrak, J. Luo, D. Zeeuwe, A. Eiben, The influence of robot traits and evolutionary dynamics on the reality gap, *IEEE Trans. Cogn. Dev. Syst.* (2021).
- [71] S. Kriegman, A.M. Nasab, D. Shah, H. Steele, G. Branin, M. Levin, J. Bongard, R. Kramer-Bottiglio, Scalable sim-to-real transfer of soft robot designs, in: *2020 3rd IEEE International Conference on Soft Robotics, RoboSoft, IEEE*, 2020, pp. 359–366.
- [72] S. Kriegman, D. Blackiston, M. Levin, J. Bongard, A scalable pipeline for designing reconfigurable organisms, *Proc. Nat. Acad. Sci.* 117 (2020) 1853–1859.
- [73] E. Hupkes, M. Jelisavcic, A.E. Eiben, Revolve: A versatile simulator for online robot evolution, in: *Applications of Evolutionary Computation: 21st International Conference, EvoApplications 2018, Parma, Italy, April 4–6, 2018, Proceedings 21, Springer*, 2018, pp. 687–702.
- [74] F. Veenstra, K. Glette, How different encodings affect performance and diversification when evolving the morphology and control of 2D virtual creatures, in: *The 2020 Conference on Artificial Life, ALIFE 2020, MIT Press*, 2020, pp. 592–601.