A Fully-distributed Shape-aware Neural Controller for Modular Robots

Giorgia Nadizar
DMG - University of Trieste
Trieste, Italy
giorgia.nadizar@phd.units.it

Eric Medvet
DIA - University of Trieste
Trieste, Italy
emedvet@units.it

Kathryn Walker
IT University Copenhagen
Copenhagen, Denmark
kwal@itu.dk

Sebastian Risi
IT University Copenhagen
Copenhagen, Denmark
sebr@itu.dk

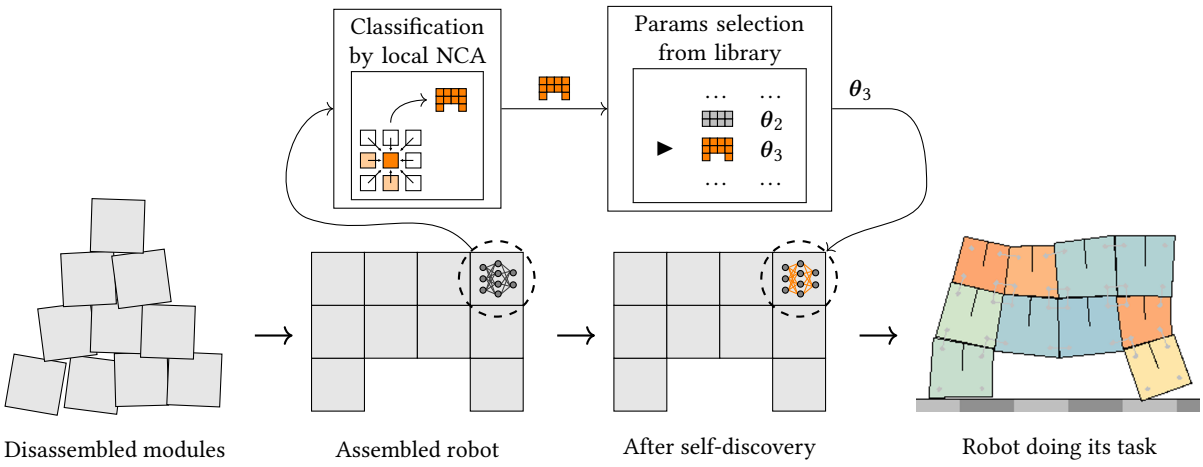

Figure 1: Overview of the proposed approach. First, modules are assembled (or auto-assemble) into a shape. Then, the NCA inside each module starts the self-discovery phase, aimed at recognizing the shape of the assembly, by exchanging information with adjacent modules. When the NCA has reached a decision on the shape it believes to be part of, the module selects accordingly the neural controller parameters $\boldsymbol{\theta}$ to be used, from a library of pre-evolved vectors of parameters. Finally, the robot performs its task, with each module using its own $\boldsymbol{\theta}$ selected in the self-discovery phase. In all phases, the modules are completely independent, but act collectively for achieving a common goal.

ABSTRACT

Modular robots are promising for their versatility and large design freedom. Modularity can also enable automatic assembly and reconfiguration, be it autonomous or via external machinery. However, these procedures are error-prone and often result in misassemblings. This, in turn, can cause catastrophic effects on the robot functionality, as the controller deployed in each module is optimized for a different robot shape than the actual one. In this work, we address such shortcoming by proposing a shape-aware modular controller, operating with (1) a *self-discovery* phase, in which each module controller identifies the shape it is assembled in, followed by (2) a *parameter selection* phase, where the controller selects its parameters according to the inferred shape. We deploy a self-classifying neural cellular automaton for phase (1), and we leverage evolutionary optimization for implementing a library of controller parameters for phase (2). We test the validity of the proposed method considering voxel-based soft robots, a class of modular soft robots, and the task of locomotion. Our findings confirm the effectiveness of such a controller paradigm, and also show that it can be used to partially overcome unforeseen damages or assembly mistakes.

GECCO '23, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0119-1/23/07…$15.00
https://doi.org/10.1145/3583131.3590419

CCS CONCEPTS

• **Computing methodologies** → *Evolutionary robotics*; **Mobile agents**; • **Computer systems organization** → *Neural networks.*

KEYWORDS

Evolutionary Robotics, Neural Cellular Automata, Embodied Intelligence, Neuroevolution, Collective Intelligence

ACM Reference Format:
Giorgia Nadizar, Eric Medvet, Kathryn Walker, and Sebastian Risi. 2023. A Fully-distributed Shape-aware Neural Controller for Modular Robots. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3583131.3590419

1 INTRODUCTION

Being able to determine one's shape would be an incredibly useful ability for many robot applications. Especially modular robots, whose configuration might be changed voluntarily or through damage, could benefit from knowing their overall composition, either

to repair damage, or to determine which compensatory behavior to execute.

While previous approaches have allowed robots to adapt by learning a repository of behaviors that could be used in response to damage [20], they did not try to determine the shape of the robot—which could speed up adaptation—but instead relied on an iterative trial-and-error approach. The work by Bongard et al. [4] explored enabling a robot to create a model of itself, but relied on actuating the robot to determine its morphology; depending on the type of damage or robot, self-modeling through actuation-sensation might not always be feasible.

Building on the ability of many biological systems that can correctly determine their own anatomically structure solely through the local interaction of cells, we present an approach that allows modular soft robots to infer their shape without the need for actuation or trial-and-error methods. Our approach builds on recent work in shape classification through neural cellular automata (NCA) [26, 31] and in collective control of modular robots [8, 22, 25]. Every module in the robot is running a copy of the same NCA that tries to determine which morphology it is part of through local communication. Based on the inferred shape, every module then sets the parameters of an embedded neural controller and thus the behavior that should be executed.

Our results demonstrate that: (a) NCA are able to accurately infer the shape of different soft robots that they are a part of, (b) this ability can be used to efficiently select controller parameters from a pre-evolved library that allow the robots to locomote, and (c) the approach enables our modular soft robots to deal with assembly inaccuracies and online damages. Because of its completely decentralized nature, the approach naturally lends itself to being deployed in modular robots in the real world in the future, opening up interesting new possibilities for more resilient physical robots.

2 RELATED WORKS

In the recent years, there has been increasing interest in making robots more resilient [33], with the eventual aim of making them more useful in practice. Resiliency can be pursued in many ways, ranging from self-healing materials [3, 14] to the robots ability to recover from damage. Damage recovery, in turn, can be achieved by trying to adapt the body of the robot (e.g., by regrowing damaged parts) or, as done in this paper, by using compensatory behaviors. We discuss related works relevant to both approaches.

Shah et al. [28] provided an overview on methods and applications related to the ability of a robot to change its shape. There are a few works in which shape change has been used to cope with damages. Kriegman et al. [13] considered voxel-based soft robots and faced damages changing the relative size and shape of voxels. Horibe et al. [11], instead, used NCA to regrow body parts when damaged, working with the same kind of robots. However, the morphological adaption approach to damage recovery has its limitations, especially when considering real world applications. Indeed, current robots lack the ability to regrow parts to overcome damages.

The alternative approach is to adapt the controller of the robot in response to morphological damage. In [4], a four legged robot infers its own structure through a sensor system and then uses this self model to generate a controller for forward locomotion. Therefore, when the robot is damaged, it is capable of creating a new self model and generating a new controller. In [5] the original controller of the robot is altered by an "intelligent trial-and-error" algorithm until a new successful controller is found for use on the damaged robot. In contrast to the previously mentioned works, our approach is completely decentralized. In principle, decentralization, together with the ability to re-use exhausted robot modules and auto-fabrication, is an enabling factor for an ecosystem of robots that adapt over time [9].

3 BACKGROUND

3.1 Voxel-based Soft Robots

We deal with voxel-based soft robots (VSRs), a kind of modular robots composed of several soft elastic cubes (voxels) which are rigidly linked together in a predefined shape. Namely, we consider the 2-D variant of simulated VSRs proposed in [16] that attempts to mimic their real fabricable counterpart, initially devised by Hiller and Lipson [10]. In this 2-D variant, VSRs are composed of equally sized squares, rather than cubes. Their simulation is performed in discrete time and continuous space.

The movement of a VSR is the result of the change of the area of its voxels over time. In turn, the area of each voxel changes as the consequence of (a) external forces acting on the voxel, namely, gravity and reaction forces deriving from the contact with other bodies, and (b) an internal force dictated by an actuation value. More precisely, we denote by $a^{(k)} \in [-1, 1]$ the *actuation value* for a voxel at time step k, with -1 corresponding to the maximum dictated expansion and 1 corresponding to the maximum dictated contraction. In practice, the simulator models contraction and expansion of the voxels as instantaneous changes of the rest lengths of several spring-damper systems composing their structure; at the same time, spring-damper systems confer softness to the voxels. We refer the reader to [16] for further details about the mechanical models of the VSR.

Formally, a VSR is defined by a shape and a controller. The *shape* is a Boolean $w \times h$ grid in which the element at cell x, y is true if and only if there is a voxel at the corresponding position. The *controller* determines the actuation value $a^{(k)}_{x,y}$ for each voxel of the VSR at each time step k.

In this study, we use the distributed neural controller proposed in [15] and later refined in [22]. For each voxel, it determines the actuation value based on some sensory inputs acquired at the voxel and some communication values coming from the adjacent voxels, which generated them at the previous time step. The processing of inputs and communication values is performed by a feed-forward artificial neural network (NN) embedded in the voxel.

In detail, at each time step k and for each voxel at (x, y) we first take the local *sensor reading* $\boldsymbol{r}^{(k)}_{x,y}$ and the *communication values* computed at the previous time step $k-1$ by the adjacent voxels $c^{(k-1)}_{x,y-1}$, $c^{(k-1)}_{x,y+1}$, $c^{(k-1)}_{x+1,y}$, and $c^{(k-1)}_{x+1,y}$. Then, we use the NN to compute the actuation value $a^{(k)}_{x,y}$ and the communication value $c^{(k)}_{x,y}$ to be passed to adjacent voxels at the next time step as:

$$\left[a^{(k)}_{x,y}\ c^{(k)}_{x,y}\right] = \mathrm{NN}_{\boldsymbol{\theta}}\left(\left[\boldsymbol{r}^{(k)}_{x,y}\ c^{(k-1)}_{x,y-1}\ c^{(k-1)}_{x,y+1}\ c^{(k-1)}_{x+1,y}\ c^{(k-1)}_{x+1,y}\right]\right), \quad (1)$$

where $\boldsymbol{\theta} \in \mathbb{R}^p$ is the vector of the parameters (or weights) of the NN. If no adjacent voxel is present in a given direction, we set the corresponding communication value to 0 in the NN input.

We use the same NN architecture, the same activation function (tanh), and the same parameters $\boldsymbol{\theta}$ in each voxel. Precisely, we use a single hidden layer of 5 neurons that, between the input layer of 8 neurons and the output layer of 2 neurons, results in an NN with $p = |\boldsymbol{\theta}| = (8+1)\cdot 5 + (5+1)\cdot 2 = 57$ weights.

The local sensor reading $\boldsymbol{r}_{x,y}^{(k)} \in [-1,1]^4$ consists of: (a) the ratio between the current area of the voxel and its rest area, (b) the distance to the closest body along the negative y-axis (i.e., "below" the voxel, integral with the voxel), and (c) the velocities of the center of mass of the voxel along the x- and y-axes (integral with the voxel). For the area ratio, we first clip the value in $[0.5, 1.5]$, then we normalize it to $[-1, 1]$. For the distance, we first cap it to 0.5, such that the raw sensed value is in $[0, 0.5]$, then we normalize it to $[-1, 1]$; we remark that other voxels of the VSR are sensed, i.e., voxels with another voxel attached below always read a distance of 0 corresponding to a normalized value of -1. For the velocities, we first clip them to $[-10, 10]$, then we normalize them to $[-1, 1]$. For reference, the side of the voxel is 1 m and the velocity range is expressed in $\mathrm{m\,s^{-1}}$.

Despite its apparent simplicity, in particular, despite adopting the same NN in each voxel, the distributed neural controller has been shown to be effective for controlling VSRs, also in the challenging case where they change shape over their life [21]. In practice, the exchange of communication values among voxels with a delay of one time step makes the compound of all the NNs a recurrent NN, which, with the proper parameters, can generate rich dynamics and, hence, effective behaviors. However, it turns out that such richness often results, upon optimization, in vibrating behaviors [17] that would be poorly effective in reality—an instance of the so-called reality gap problem [20, 27, 30]. For this reason, we actually compute the actuation values at a lower frequency than the simulation frequency, namely at 5 Hz rather than 60 Hz, as done in [19]; in between computations, we apply the lastly computed actuation value for each voxel.

A peculiarity of the distributed neural controller, which we exploit in this study, is that it realizes a form of collective intelligence [22]: the overall behavior of the VSR results (also) from the communications exchanged between the NNs embedded in the voxels which are, however, independent from each other. In practice, one can replace one NN in a voxel with a different NN with the same input and output size and the VSR is still functional, i.e., it can behave, tough its behavior might be different from the one obtained with the "original" NN. As we will see in Section 4.3, we exploit this possibility to inject in some voxels the NNs that have been optimized for a different shape than the one the affected voxels belong to.

3.2 Neural cellular automata

The shape detection part of our system is based on the concept of cellular automata (CA). CA consist of "cells" arranged into a grid. Each cell state updates according to the states of the neighboring cells and a set of rules, the same for all the cells within the grid. A classic example of CA is Conway's game of life: in this simple

Set	Morph.	$w\times h$	n	$v_x^\star$	g_{80}	Evol.
C_1, C_2, C_3, C_4		4×3	10	3.5 ± 0.2	29 ± 25	
C_1, C_2, C_3, C_4		5×2	10	1.0 ± 0.5	37 ± 37	
C_1, C_2, C_3, C_4		7×2	11	1.9 ± 0.2	30 ± 15	
C_2, C_4		4×2	6	2.2 ± 0.1	43 ± 16	
C_2, C_4		4×1	4	0.7 ± 0.2	14 ± 39	
C_2, C_4		5×2	8	1.5 ± 0.1	34 ± 7	
C_2, C_4		6×4	20	2.2 ± 0.3	54 ± 24	
C_2, C_4		8×3	24	1.4 ± 0.3	36 ± 27	
C_2, C_4		9×3	23	2.3 ± 0.7	31 ± 3	
C_3, C_4		4×3	9	3.9 ± 0.2	26 ± 8	
C_3, C_4		4×3	9	3.6 ± 0.3	31 ± 9	
C_3, C_4		4×3	9	3.7 ± 0.2	27 ± 4	
C_3, C_4		4×3	9	2.2 ± 0.3	79 ± 22	
C_3, C_4		5×2	9	0.9 ± 0.7	20 ± 33	
C_3, C_4		5×2	9	1.7 ± 0.5	29 ± 16	
C_3, C_4		5×2	9	1.6 ± 0.3	42 ± 28	
C_3, C_4		5×2	9	1.2 ± 0.2	48 ± 50	
C_3, C_4		7×2	10	1.9 ± 0.4	47 ± 20	
C_3, C_4		7×2	10	1.5 ± 0.5	56 ± 27	
C_3, C_4		7×2	10	2.2 ± 0.4	16 ± 9	
C_3, C_4		7×2	10	1.7 ± 0.4	55 ± 25	
C_4		4×2	5	3.5 ± 0.1	17 ± 5	
C_4		4×2	5	1.3 ± 0.6	82 ± 32	
C_4		3×1	3	0.3 ± 0.1	15 ± 13	
C_4		5×2	7	1.8 ± 0.1	32 ± 14	
C_4		5×2	7	2.2 ± 1.0	47 ± 19	
C_4		5×2	7	1.6 ± 0.3	105 ± 53	
C_4		6×4	19	2.5 ± 0.5	55 ± 11	
C_4		6×4	19	2.1 ± 0.3	41 ± 31	
C_4		6×4	19	3.0 ± 0.1	30 ± 7	
C_4		6×4	19	2.3 ± 0.6	58 ± 16	
C_4		8×3	23	1.5 ± 0.1	28 ± 17	
C_4		8×3	23	1.8 ± 0.1	49 ± 22	
C_4		8×3	23	1.6 ± 0.2	33 ± 15	
C_4		8×3	23	0.9 ± 0.3	12 ± 27	
C_4		9×3	22	2.7 ± 0.6	25 ± 14	
C_4		9×3	22	2.5 ± 0.4	35 ± 10	
C_4		9×3	22	2.3 ± 0.2	33 ± 18	
C_4		9×3	22	2.7 ± 0.1	20 ± 7	

Table 1: Salient information about the VSR shapes and the outcome of the evolutionary optimization of their controllers. The rightmost column shows an overview of the evolution: the orange line shows the fitness v_x of the best individual along generations; for reference, the gray line shows its average across all shapes. For shapes derived from others, i.e., those belonging to sets $C_3 \setminus C_1$ and $C_4 \setminus C_2$, the green line shows the evolution of v_x for the original shape (e.g.,  derives from ).

example, the state of each cell is a Boolean and is updated according to four simple rules.

Neural cellular automata (NCA) are an extension of CA. Whilst the cells are still arranged within a grid, the state of a cell can now be represented by a real valued vector. Instead of simple hand coded rules, the rules governing cell updates are functions parameterized by an NN [24, 26, 31].

Recently, NCA have been used for performing shape detection or classification. Two approaches that are particularly relevant for this work are those of Randazzo et al. [26] and of Walker et al. [31]. In the former, NCA are trained to be capable of self-classifying MNIST digits (handwritten numbers) irrespective of the digit shape or size. In the latter, which is the main inspiration behind this work, NCA are trained for 2-D shape classification, and are also physically realized in a 2-D static robotic system.

4 EVOLVING SHAPE-AWARE CONTROLLERS

The *shape-aware* controller is composed of three core elements: (1) an NCA for shape self-classification, (2) a library of controller parameters realized with evolutionary optimization, and (3) the actual controller which combines the former two. We detail the internals of each component in the following.

4.1 NCA for shape self-classification

For the shape detection component we rely on the approach to *self-classification* proposed by Walker et al. [31], based on NCA. In this framework, the classification is the result of the update over time of the cell state, thanks to an update function and the information shared with neighbors. In turn, the NCA cell state plays a two-fold role, storing information to be passed to neighbors, and encoding the current guess w.r.t. the self-classification.

More in detail, we represent the state of each NCA cell as a real-valued vector of size n_s. We initialize the state vector of each cell at x, y with zeros, $\boldsymbol{s}_{x,y}^{(0)} = \mathbf{0} \in \mathbb{R}^{n_s}$, with the exclusion of the first component, which we set to 1 to indicate the presence of an "active" NCA cell. Then, we iteratively update the state of the cell as:

$$\boldsymbol{s}_{x,y}^{(h)} = \boldsymbol{s}_{x,y}^{(h-1)} + \mathrm{CNN}_{\boldsymbol{v}}\left(\begin{bmatrix} \mathbf{0} & \boldsymbol{s}_{x,y-1}^{(h-1)} & \mathbf{0} \\ \boldsymbol{s}_{x-1,y}^{(h-1)} & \boldsymbol{s}_{x,y}^{(h-1)} & \boldsymbol{s}_{x+1,y}^{(h-1)} \\ \mathbf{0} & \boldsymbol{s}_{x,y+1}^{(h-1)} & \mathbf{0} \end{bmatrix}\right), \tag{2}$$

where h indicates the update time step and $\mathrm{CNN}_{\boldsymbol{v}}$ is the update function, a convolutional neural network (CNN) parametrized by $\boldsymbol{v}$. Here, we employ the Von Neumann neighborhood, i.e., the 4 directly adjacent neighbors, and the cell itself as input to the CNN, padding the corner elements with zero vectors $\mathbf{0} \in \mathbb{R}^{n_s}$ to obtain a $3 \times 3 \times n_s$ tensor. We use the same padding technique in case of missing neighbors, i.e., on the borders of the shape.

We adopt a CNN with 3 layers: the first layer uses a 3×3 convolution with $n_{c,0}$ channels and a Rectified Linear Unit (ReLU) activation function, the second layer employs a single 1×1 convolution with $n_{c,1} = n_{c,0}$ channels and ReLU, while in the last layer we have a linear 1×1 convolution with $n_{c,2} = n_s$ channels, as the amount of channels in the output layer needs to match the cell state size n_s.

For the classification task, we consider the arg max over the last $|C|$ channels of the cells state, where $|C| < n_s$ is the cardinality of the set of shapes C, i.e., classes, employed. The arg max holds the (index of the) shape predicted by the NCA. Clearly, the inference, i.e., the classification, phase in the self-classifying NCA is guided by the cells state updates over time. Thus, it cannot be performed instantaneously, but actually requires a certain amount of update steps to be performed to allow enough information exchange between cells.

We train the self-classifying NCA, i.e., we optimize its parameters $\boldsymbol{v}$, using Stochastic Gradient Descent (SGD) and the Adam optimizer [12]. We aim at minimizing a loss defined as

$$\mathcal{L}_{\mathrm{NCA}}(v) = \sum_{j=1}^{|C|} \sum_{h=h_i}^{h_f} \sum_{(x,y)\in c_j} \frac{1}{|c_j|} \left\| \boldsymbol{s}_{\mathrm{last},x,y}^{(h)} - \boldsymbol{c}_j \right\|^2, \tag{3}$$

where $|C|$ is the number of shapes employed, h_i and h_f are the initial and final update steps considered, respectively, c_j is the set of the j-th shape cells coordinates, $\boldsymbol{s}_{\mathrm{last},x,y}^{(h)}$ is the vector of the last $|C|$ elements of the vector state $\boldsymbol{s}_{x,y}^{(h)}$ of cell at x, y at time step h, $\boldsymbol{c}_j$ is the one-hot encoded class label, and $\|\cdot\|$ indicates the Euclidean norm. We take inspiration from [31] for the loss definition. However, we slightly alter the original loss concerning the number of updates: instead of applying a random number of update steps and taking the final classification outcome, here we update the NCA for a fixed amount of steps h_f, and we consider all the outcomes achieved starting from h_i steps. This way we promote convergence and stability in the NCA.

4.2 Controller library evolution

As seen in Section 3.1, we rely on NNs inserted within voxels to control VSRs. When we handcraft a VSR by assembling individual voxels into a given shape, we equip all of them with the same NN parameters $\boldsymbol{\theta}$. Clearly, it is advisable to use the parameters $\boldsymbol{\theta}_i^\star$ which best control the ensemble of voxels assembled in a certain shape c_i. Here, given a set of shapes, we want to draw a correspondence from each shape c_i to its optimal NN parameters $\boldsymbol{\theta}_i^\star$, practically realizing a library of optimized controllers: we will then use the NCA to tell what is the shape of the "current" robot and, hence, to choose the corresponding optimal NN parameters from the library.

For building the library, we resort to evolutionary computation for finding the optimal NN parameters $\boldsymbol{\theta}_i^\star$ for each controller, as previously done in [15] for the same case study. More in details, we rely on a simple form of evolutionary strategy (ES) for addressing the optimization problem. As we can see in Algorithm 1, we evolve a population of fixed size n_{pop} for a total of n_{gen} generations. At each generation, we select the parents as the top forth of the current population, and we compute their offspring by adding a Gaussian mutation of parameters $N(0, \sigma^2)$ to their point-wise mean. We use elitism: at each generation we retain the current best individual, while we replace all remaining individuals with the newly generated offspring.

In order to realize a controller library, we optimize $\boldsymbol{\theta}_i$ for every VSR shape c_i with Algorithm 1. To evaluate the quality of each configuration $\boldsymbol{\theta}_i$, i.e., its *fitness*, we deploy an NN controller with $\boldsymbol{\theta}_i$ in the voxels of a VSR of shape c_i, and we evaluate its degree of

function evolve():
  $P \leftarrow \emptyset$
  foreach $i \in \{1, \dots, n_{\text{pop}}\}$ **do**
    $P \leftarrow P \cup \{\mathbf{0} + U(-1, 1)^p\}$
  end
  foreach $g \in \{1, \dots, n_{\text{gen}}\}$ **do**
    $P_{\text{parents}} \leftarrow \texttt{bestIndividuals}\left(P, \left\lfloor \frac{|P|}{4} \right\rfloor\right)$
    $\boldsymbol{\mu} \leftarrow \texttt{mean}(P_{\text{parents}})$
    $P' \leftarrow \{\texttt{bestIndividuals}(P, 1)\}$
    while $|P'| < n_{\text{pop}}$ **do**
      $P' \leftarrow P' \cup \{\boldsymbol{\mu} + N(0, \sigma^2)^p\}$
    end
    $P \leftarrow P'$
  end
  return $\texttt{bestIndividuals}(P, 1)$
end

Algorithm 1: The simple ES used for optimizing the NN parameters for each shape.

accomplishment of a certain task. Here we consider locomotion as task, where the goal of the VSR is to move as fast as possible along the positive x direction. Hence, we use the velocity v_x measured in one simulation as the fitness of the agent, which we aim at maximizing. Trivial though it may seem, this task is the most common one in evolutionary robotics, and it is used as it can be a simple yet general enough task to evaluate an artificial agent.

4.3 Shape-aware controller

The actual shape-aware controller combines the elements described in Sections 4.1 and 4.2 to recognize the shape of the VSR and select the most appropriate parameters for it. Namely, we can distinguish two operating phases: (1) the self-classification with the NCA and (2) the controller parameters selection from the library. Once the controller parameters are selected, the controller determines the robot behavior that attempts to accomplish the given task.

The initial phase relies of the self-classifying NCA. Right after assembly, the NCA within all modules synchronously update for a total of n_u steps according to Equation (2). This phase could be triggered by an external signal to ensure proper synchronization among voxels, although synchronism and determinism could also be sacrificed without significant performance loss, as seen in [31]. After n_u update steps, every NCA has self-classified the shape it belongs to, and the outcome can in principle differ from voxel to voxel.

At this point, the parameter update phase starts. For each voxel the outcome of the local classification into shape c_i determines the parameters $\boldsymbol{\theta}_i^\star$ to be chosen from the previously optimized library. Hence, the controller for said voxel becomes $\text{NN}_{\boldsymbol{\theta}_i^\star}$, and the voxel within the VSR immediately starts its actions under its control.

We remark that since the classification outcomes may differ within the assembled VSR, also the controllers deployed in individual voxels may be different. As this is often not a desirable condition, it is possible to add an optional *majority voting* phase prior to the controller selection, to ensure global agreement and homogeneity. The majority voting remains fully compatible with modularity: in fact, several algorithms for reaching distributed consensus exist [32]. Here we provide a simple implementation which only requires each voxel to have a unique identifier and to store and propagate its vote and all the received ones relying on a table (id, v), id being the voxel unique identifier and v the corresponding vote. At each time step, such table is passed on to the neighbors, and is updated merging it with the ones received by adjacent voxels. This takes $O(n)$ steps to converge, n being the amount of voxels involved, and can be automatically stopped either when the tables are not updated anymore or with a external trigger.

5 EXPERIMENTAL EVALUATION

In our experimental evaluation we aimed at addressing the following research questions:

- RQ1 Is the shape-aware controller effective? Does its effectiveness vary with the amount of shapes included in the library?
- RQ2 How does the effectiveness of the controller depend on the NCA hyper-parameters? Is it possible to shrink the CNN in the NCA within it while retaining its original capability? Is it robust w.r.t. the number of state updates performed for self-classification?
- RQ3 Is the shape-aware controller a viable approach to tackle unforeseen assembly inaccuracies? In other words, what happens if the final shape does not belong to the controllers library?

For answering these questions, we performed several experiments in different conditions. Namely, we considered a total of 39 VSR shapes, which we organized into 4 sets of increasing size for training the self-classifying NCA. We evaluated the performance of the shape-aware controller by comparing the results it achieved with those obtained with the correct controller for each shape. We also repeated our assessment changing the NCA parameters, namely the CNN size and the amount of update steps. Last, we tested the behavior of the shape-aware controller on unseen shapes to assess its applicability for reacting to unforeseen mistakes.

5.1 RQ1: is the shape-aware controller effective?

To provide an answer to the first research question, i.e., to test the effectiveness of the proposed controller, we started by assessing the performance of its building blocks. Namely, we tested whether the self-classifying NCA is accurate on VSR shapes, and if all the considered shapes are suitable for evolving a controller for the task of locomotion.

Concerning the NCA component, we followed the training procedure described in Section 4.1. We considered 4 sets of shapes of increasing sizes ($|C_1| = 3$, $|C_2| = 9$, $|C_3| = 15$, $|C_4| = 39$, see Table 1 for the shapes in each set) to evaluate the dependency of the performance of the NCA w.r.t. the shapes involved, i.e., w.r.t. the difficulty of the classification task. For each set we trained an NCA for self-classification for 1500 iterations with SGD and the Adam optimizer with default parameters, employing TensorFlow as software tool [1]. We repeated the training 10 independent times for each configuration, to ensure results were not just a result of randomness. Regarding the NCA parameters, we fixed $n_{c,0} = n_{c,1} = 80$ for all the NCA, while we set $n_{c,2} = |C_i| + 20$, where $|C_i|$ is the

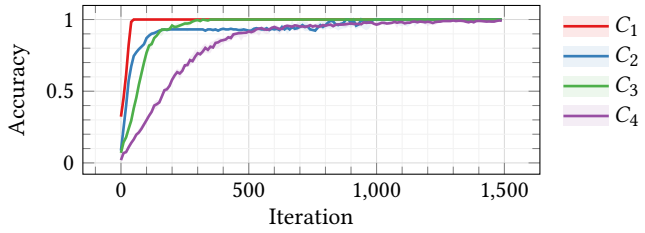

Figure 2: Accuracy during training for different shapes sets (median and inter-quartile range across 10 trainings) .

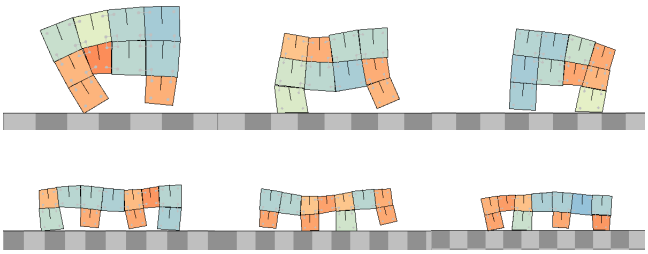

Figure 3: Frames of VSRs with the shape-aware controller successfully performing locomotion.

size of the considered set. Moreover, we computed the loss $\mathcal{L}_{\text{NCA}}$ considering $h_i = 25$ and $h_f = 50$.

We display a summary of the NCA training results in Figure 2. For each set, we report the progression of the accuracy (averaged across shapes, cells, and time) achieved at training time along iterations, in terms of median and inter-quartile range across the 10 trainings. From the plots we can note a clear convergence towards the perfect accuracy (i.e., 1) for all sets, with different slopes dictated by the difficulty of each set. Hence, we can conclude that the NCA is indeed capable of performing self-classification on the considered VSR shapes. An interesting remark can be made about set 2, which displays slower convergence w.r.t. the larger set 3. We speculate this derives from the great shape size variability within set 2, which makes the classification task harder regardless of the more contained size of the set.

After testing the capability of the NCA, we moved on to assessing the possibility of implementing a library of controllers optimized for each of the considered shapes. To this end, we followed the evolutionary optimization procedure described in Section 4.2 for each of the shapes of Table 1. We recall that we built each set of shapes that origins the library by hand, to cover a diverse enough variety of shapes, although it is also possible to frame the shapes choice as an optimization problem itself, as in [17, 29]. Concerning the controller, we used a distributed neural controller, consisting of the same $\text{NN}_{\boldsymbol{\theta}}$ in each voxel. We optimized $\boldsymbol{\theta}_i$ for each shape c_i for the task of locomotion on a flat terrain, i.e., maximizing v_x, considering a simulation of 30 s, from which we discarded the initial transient of 5 s. We relied on the ES presented in Algorithm 1, with $\sigma = 0.35$, $n_{\text{pop}} = 48$, and $n_{\text{gen}} = 208$ (corresponding to $\approx 10\,000$ total fitness evaluations). For each of the considered shapes we repeated the evolutionary optimization 5 independent times, thus achieving 5 optimized controller libraries. As software tools we utilized 2D-VSR-Sim [16] for the VSR simulation and JGEA [18] for the evolutionary optimization.

The controller optimization results are summarized in Table 1. For each shape we report the median and the standard deviation across the 5 optimizations of the velocity of the best individual at the end of evolution $v_x^\star$, and its trend along evolution in the last column. Moreover, we quantify how easy and fast it is to optimize a shape with g_{80}, that is the amount of generations needed to reach 80 % of the performance achieved at the end of evolution. From the table we clearly see that some shapes are more effective than others for the locomotion task, i.e., some achieve higher $v_x^\star$, and we also notice that it takes different efforts, i.e., different g_{80}, to optimize some of them. However, we observe that for all shapes there exists a controller $\text{NN}_{\boldsymbol{\theta}}$ which leads them to achieve an effective locomotion gait. We also verified this by visually inspecting the videos[1] of the whole optimized library of shape-controller pairs for one of the independent optimizations—a sample of behaviors is shown in Figure 3.

Having tested the effectiveness of both building blocks, we proceeded to evaluating their combination, following the methodology described in Section 4.3. Namely, for each shape in each set, we proceeded as follows: (1) we assembled a VSR in the given shape, (2) we let the self-classifying NCA update for $n_u = 40$ steps, and (3) we used the outcome of the shape classification to select the controller for the VSR. We set $n_u = 40$ as a reasonable value in-between $h_i = 25$ and $h_f = 50$, considered at training time. We repeated the evaluation for each set, for each of the 10 trained NCA, and for each of the 5 controller libraries. In addition, we repeated the procedure introducing the majority voting phase prior to the controller selection, as seen in Section 4.3.

As evaluation metrics we considered the NCA *classification accuracy*, defined as the fraction of correctly classified voxels in the VSR, and the *relative performance loss* $\rho = 1 - \frac{v}{v_x^\star}$, v being the velocity achieved in a new simulation with the NCA-selected controller. We report the results obtained for all the sets in Table 2, in terms of median and standard deviation. The most outstanding trait of the table is that the results achieved in terms of median are always "perfect", meaning that the majority of tests led to an ideal result. However, the non-zero standard deviation for sets C_2 and C_4 hints that in some cases mis-classifications occurred, which led to non-negligible performance losses. Another interesting observation regards the effects of majority voting: looking at sets C_2 and C_4 we see that it often leads to a decrease in the standard deviation, meaning that it can help to cope with minor classification errors. Therefore, we can conclude that such additional layer of complexity could be worth it to deal with more complex scenarios, while it becomes practically useless is easier conditions.

[1] available at https://drive.google.com/drive/folders/12t2Wt07PzQCRFTkeEwCi4rK_71F9S650?usp=sharing

	W/o majority voting		W/ majority voting	
Set	Accuracy	ρ	Accuracy	ρ
C_1	1.00	0.00	1.00	0.00
C_2	1.00±0.21	0.00±0.21	1.00±0.23	0.00±0.18
C_3	1.00	0.00	1.00	0.00
C_4	1.00±0.17	0.00±0.20	1.00±0.17	0.00±0.13

Table 2: Median and standard deviation of accuracy and relative performance loss ρ (with and without majority voting) after 40 developmental steps of the NCA for each set.

	W/o majority voting		W/ majority voting	
Set	Accuracy	ρ	Accuracy	ρ
C_1	1.00	0.00	1.00	0.00
C_2	1.00±0.36	0.00±0.37	1.00±0.38	0.00±0.29
C_3	1.00±0.16	0.00±0.27	1.00±0.12	0.00±0.11
C_4	0.67±0.44	0.55±0.46	1.00±0.50	0.00±0.44

Table 3: Median and standard deviation of accuracy and relative performance loss ρ (with and without majority voting) after 40 developmental steps using a smaller CNN to govern the NCA.

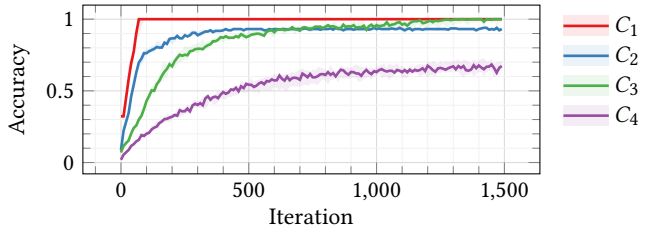

Figure 4: Accuracy during training for different shapes sets using a smaller CNN to govern the NCA (median and inter-quartile range across 10 trainings).

5.2 RQ2: how do the NCA parameters affect the controller effectiveness?

To evaluate the impact of the NCA parameters on the overall effectiveness of the shape-aware controller, we reproduced the experimental evaluation described in Section 5.1 with different parameters. Namely, we performed a two-fold study, to assess the impact of the size of the CNN in the NCA, and the robustness w.r.t. the amount of update steps n_u employed for the self-classification.

5.2.1 CNN size. First, we aimed at assessing the impact of a smaller NCA architecture, as in physical implementations it might be desirable to shrink the CNN while retaining good performance [23]. To test the feasibility of this idea, we repeated the experimental procedure of Section 5.1 with a drastically reduced CNN within the NCA. Namely, we set $n_{c,0} = n_{c,1} = 30$ and $n_{c,2} = |C_i| + 10$, $|C_i|$ being the size of the considered set.

We report the accuracy results achieved at training time in Figure 4 for all the considered sets. Comparing them with those of Figure 2 we clearly notice that, although the trends are similar, the size of the CNN plays a key role in making the NCA capable of performing classification. In fact, here we notice good convergence only for the smaller sets, whereas for set C_4 the accuracy remains low, i.e., the NCA is not able to properly classify all the shapes.

The inaccuracies observed at training time also reflect in a non-optimal behavior when the full controller is deployed, as we can see in Table 3. Although for sets C_1 to C_3 we still observe a perfect behavior in terms of median, we notice a clear increase in the standard deviation w.r.t. Table 2. Moreover, for set C_3 we highlight a neat performance drop, if no majority voting is employed. Summarizing, a smaller CNN can suffice for easier tasks, i.e., when few shapes are employed, but it struggles when the classification becomes harder. In such cases, the majority voting scheme can be crucial for retaining a good overall controller performance.

5.2.2 NCA update steps. Another interesting aspect to evaluate regards whether the shape-aware controller is robust w.r.t. the amount of update steps performed by the NCA for the self-classification phase. To assess this, we took the NCA trained in Section 5.1, and we repeated the controller deployment phase with different values of $n_u = 1, \ldots, 100$.

We report the accuracy and the relative performance loss ρ (median and inter-quartile range) obtained for different sets, also with majority voting, for each value of n_u (on the x-axis) in Figure 5. Focusing on the results achieved with the mere NCA classification we clearly see a phase with perfect accuracy/zero ρ ranging from $n_u \approx 10$ to $n_u \approx 60$, preceded and followed by poorer controller performance. Concerning the initial phase, it naturally derives by the fact that it takes some time and some exchange of information between cells to achieve the correct classification. In fact, smaller sets display a shorter transient and perform well with fewer steps. For the final phase, instead, it could be a consequence of how the loss $\mathcal{L}_{\text{NCA}}$ is computed, as it takes into consideration only what happens, i.e., the average accuracy, between $h_i = 25$ and $h_f = 50$. Hence, we are not explicitly promoting permanent convergence when training the NCA, which results in a loss of performance after the considered interval. However, if we observe the outcome with the addition of the majority voting phase, we once again notice how it can be useful for overcoming more difficult scenarios, making up for minor mis-classifications.

The obtained results suggest that using the NCA for continuous online classification would not be possible without additional measures. In fact, although such a classification would be desirable for spotting and reacting to unexpected damages, e.g., loss of voxels during the accomplishment of a task, it appears not to be feasible due to the NCA instability at the increase of n_u. However, some simple precautions as periodic state reset or a damage detection signal could suffice for making the shape-aware controller viable for dealing with online damages. We leave the investigation of these aspects for future work.

5.3 RQ3: can the shape-aware controller deal with unforeseen inaccuracies?

In order to assess the shape-aware controller performance when reacting to unforeseen assembly inaccuracies, we performed an

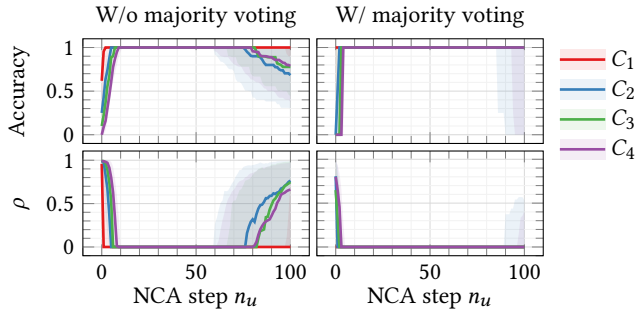

Figure 5: Accuracy and relative performance loss ρ (median and inter-quartile range) of the shape-aware controller with varying amounts of NCA update steps n_u (on the x-axis) for different sets.

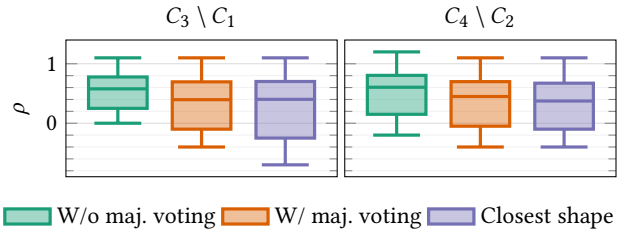

Figure 6: Distributions of the relative performance loss ρ achieved by the shape-aware controller deployed on unseen shapes. We also report the ρ obtained using the NN controller relative to the shape closest to the considered one (in terms of edit distance) in each voxel, as a baseline.

experimental study similar to that of Section 5.1, changing the set of shapes of evaluation w.r.t. that employed for training the NCA classifier. Namely, we employed set C_1 and C_2 for the training, while we relied on the larger sets C_3 and C_4, respectively, for the evaluation. Being interested in measuring the results only for unseen shapes, we removed the common shapes from the larger sets (i.e., we actually used set $C_3 \setminus C_1$ and $C_4 \setminus C_2$). In this way, we purposely considered shapes differing from the original ones for at most one missing voxel as those constitute an ideal test-bed for simulating small assembly mistakes or damages. For the controller evaluation we followed the same procedure described in Section 5.1, where we let the NCA update for $n_u = 40$ steps, and used the classification outcome to deploy the controller of each voxel. As before, we also experimented with the addition of a majority voting phase in-between classification and deployment.

As performance index we only considered the relative performance loss ρ, as it is not possible to define the accuracy (the shape is not among the ones known by the NCA). As a baseline, we also tested the performance of the VSR with all the voxels equipped with the controller relative to the closest shape among those belonging to the training set of the NCA. We report the distribution of ρ for the three aforementioned cases in the box plots of Figure 6, dividing them by the sets considered for the NCA training and assessment. Namely, on the left we display the results for the NCA trained on C_1 and assessed on $C_3 \setminus C_1$, whereas on the right we show the results relative to the training on C_2 and the assessment on $C_4 \setminus C_2$.

An interesting and surprising trait of the displayed box plots is that they extend above 1 and below 0. The first is caused by negative velocities during assessment, meaning that the VSR is moving in the wrong direction, resulting from a severely scrambled controller. The negative values of ρ, instead, are a consequence of slight performance improvements caused by a new favorable arrangement of controllers within voxels, also seen in [19].

Moving on to a comparative analysis between the box plots, we note that the baseline is usually the best performing one, displaying a generally lower ρ in comparison to the others. To further investigate on this, we performed a Mann-Whitney U test with the null hypothesis of distributions equality between each pair of box plots (the baseline, the one with the NCA classifier only, and the one with majority voting). Considering a significance level of $\alpha = 0.05$, the only pair displaying a non-significant difference is for set $C_3 \setminus C_1$ between the baseline and the controller with majority voting. Hence, we can conclude that the shape-aware controller has fairly limited generalization abilities for out-of-sample shapes. However, given the notable performance observed on even larger sets in Section 5.1, we can overcome this limitation by including a wide variety of shapes at training time, possibly encompassing all the damaged shapes versions, to make the NCA able to deal with those as well. In other words, it appears convenient to evolve, i.e., optimize, a larger library of controllers [6, 7], as this only increases the computational effort at pre-training time, while notably enhancing the deployed controller performance, possibly also when reacting to online damages [2].

6 CONCLUSION

In this work, we proposed a shape-aware neural controller for modular robots, which can be embedded within individual modules and only requires communication with direct neighbors to function. Here shape-aware refers to each module controller's ability to automatically infer the overall shape of the agent.

To validate the approach, we conducted a thorough experimental evaluation on a locomotion task with simulated voxel-based soft robots, a class of modular robots composed of elastic cubes. Our results demonstrate the effectiveness of the shape-aware controller, and confirm it can effectively rely on self-classification to select the proper parameters to use among a pre-optimized library, obtained with evolutionary computation. Moreover, we show its applicability to different scenarios, such as dealing with assembly inaccuracies or online damages.

We believe our approach paves the way for an autonomous robot ecosystem, where automatic assembly can be followed by self-discovery and autonomous controller selection and deployment. Although we conducted this study in simulation only, the completely decentralized approach should make it amendable for real-world transfer. Thus, in the future we aim to explore the applicability of the proposed approach for physical modular robots.

ACKNOWLEDGMENTS

This work was partially supported by a Sapere Aude: DFF Starting Grant (9063-00046B).

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 265–283.

[2] Maxime Allard, Simón C Smith, Konstantinos Chatzilygeroudis, Bryan Lim, and Antoine Cully. 2022. Online Damage Recovery for Physical Robots with Hierarchical Quality-Diversity. *arXiv preprint arXiv:2210.09918* (2022).

[3] R Adam Bilodeau and Rebecca K Kramer. 2017. Self-healing and damage resilience for soft robotics: A review. *Frontiers in Robotics and AI* 4 (2017), 48.

[4] Josh Bongard, Victor Zykov, and Hod Lipson. 2006. Resilient machines through continuous self-modeling. *Science* 314, 5802 (2006), 1118–1121.

[5] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. 2015. Robots that can adapt like animals. *Nature* 521, 7553 (2015), 503–507.

[6] Antoine Cully and Jean-Baptiste Mouret. 2016. Evolving a behavioral repertoire for a walking robot. *Evolutionary computation* 24, 1 (2016), 59–88.

[7] Miguel Duarte, Jorge Gomes, Sancho Moura Oliveira, and Anders Lyhne Christensen. 2017. Evolution of repertoire-based control for robots with complex locomotor systems. *IEEE Transactions on Evolutionary Computation* 22, 2 (2017), 314–328.

[8] David Ha and Yujin Tang. 2022. Collective intelligence for deep learning: A survey of recent developments. *Collective Intelligence* 1, 1 (2022), 26339137221114874.

[9] Matthew F Hale, Edgar Buchanan, Alan F Winfield, Jon Timmis, Emma Hart, Agoston E Eiben, Mike Angus, Frank Veenstra, Wei Li, Robert Woolley, et al. 2019. The ARE Robot Fabricator: How to (Re) produce Robots that Can Evolve in the Real World. In *Artificial Life Conference Proceedings*. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info ..., 95–102.

[10] Jonathan Hiller and Hod Lipson. 2012. Automatic design and manufacture of soft robots. *IEEE Transactions on Robotics* 28, 2 (2012), 457–466.

[11] Kazuya Horibe, Kathryn Walker, Rasmus Berg Palm, Shyam Sudhakaran, and Sebastian Risi. 2022. Severe damage recovery in evolving soft robots through differentiable programming. *Genetic Programming and Evolvable Machines* 23, 3 (2022), 405–426.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Sam Kriegman, Stephanie Walker, Dylan Shah, Michael Levin, Rebecca Kramer-Bottiglio, and Josh Bongard. 2019. Automated shapeshifting for function recovery in damaged robots. *arXiv preprint arXiv:1905.09264* (2019).

[14] Julie Legrand, Seppe Terryn, Ellen Roels, and Bram Vanderborght. 2023. Reconfigurable, multi-material, voxel-based soft robots. *IEEE Robotics and Automation Letters* (2023).

[15] Eric Medvet, Alberto Bartoli, Andrea De Lorenzo, and Giulio Fidel. 2020. Evolution of distributed neural controllers for voxel-based soft robots. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 112–120.

[16] Eric Medvet, Alberto Bartoli, Andrea De Lorenzo, and Stefano Seriani. 2020. 2D-VSR-Sim: a simulation tool for the optimization of 2-D voxel-based soft robots. *SoftwareX* 12 (2020), 100573.

[17] Eric Medvet, Alberto Bartoli, Federico Pigozzi, and Marco Rochelli. 2021. Biodiversity in evolved voxel-based soft robots. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 129–137.

[18] Eric Medvet, Giorgia Nadizar, and Luca Manzoni. 2022. JGEA: a modular java framework for experimenting with evolutionary computation. In *Proceedings of the genetic and evolutionary computation conference companion*. 2009–2018.

[19] Eric Medvet and Francesco Rusin. 2022. Impact of Morphology Variations on Evolved Neural Controllers for Modular Robots. In *XVI International Workshop on Artificial Life and Evolutionary Computation (WIVACE)*.

[20] Jean-Baptiste Mouret and Konstantinos Chatzilygeroudis. 2017. 20 years of reality gap: a few thoughts about simulators in evolutionary robotics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1121–1124.

[21] Giorgia Nadizar, Eric Medvet, and Karine Miras. 2022. On the schedule for morphological development of evolved modular soft robots. In *Genetic Programming: 25th European Conference, EuroGP 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings*. Springer, 146–161.

[22] Giorgia Nadizar, Eric Medvet, Stefano Nichele, and Sidney Pontes-Filho. 2022. Collective control of modular soft robots via embodied Spiking Neural Cellular Automata. *arXiv preprint arXiv:2204.02099* (2022).

[23] Giorgia Nadizar, Eric Medvet, Hola Huse Ramstad, Stefano Nichele, Felice Andrea Pellegrino, and Marco Zullich. 2022. Merging pruning and neuroevolution: towards robust and efficient controllers for modular soft robots. *The Knowledge Engineering Review* 37 (2022).

[24] Stefano Nichele, Mathias Berild Ose, Sebastian Risi, and Gunnar Tufte. 2017. CA-NEAT: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. *IEEE Transactions on Cognitive and Developmental Systems* 10, 3 (2017), 687–700.

[25] Federico Pigozzi, Yujin Tang, Eric Medvet, and David Ha. 2022. Evolving modular soft robots without explicit inter-module communication using local self-attention. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 148–157.

[26] Ettore Randazzo, Alexander Mordvintsev, Eyvind Niklasson, Michael Levin, and Sam Greydanus. 2020. Self-classifying mnist digits. *Distill* 5, 8 (2020), e00027–002.

[27] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. 2021. Crossing the Reality Gap: a Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning. *IEEE Access* (2021).

[28] Dylan Shah, Bilige Yang, Sam Kriegman, Michael Levin, Josh Bongard, and Rebecca Kramer-Bottiglio. 2021. Shape changing robots: bioinspiration, simulation, and physical realization. *Advanced Materials* 33, 19 (2021), 2002882.

[29] Jacopo Talamini, Eric Medvet, and Stefano Nichele. 2021. Criticality-driven evolution of adaptable morphologies of voxel-based soft-robots. *Frontiers in Robotics and AI* 8 (2021), 673156.

[30] Fuda van Diggelen, Eliseo Ferrante, Nihed Harrak, Jie Luo, Daan Zeeuwe, and AE Eiben. 2021. The influence of robot traits and evolutionary dynamics on the reality gap. *IEEE Transactions on Cognitive and Developmental Systems* (2021).

[31] Kathryn Walker, Rasmus Berg Palm, Rodrigo Moreno, Andres Faina, Kasper Stoy, and Sebastian Risi. 2022. Physical neural cellular automata for 2d shape classification. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 12667–12673.

[32] Gengrui Zhang, Fei Pan, Michael Dang'ana, Yunhao Mao, Shashank Motepalli, Shiquan Zhang, and Hans-Arno Jacobsen. 2022. Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms. *arXiv preprint arXiv:2204.03181* (2022).

[33] Tan Zhang, Wenjun Zhang, and Madan M Gupta. 2017. Resilient robots: Concept, review, and future directions. *Robotics* 6, 4 (2017), 22.