Physical Neural Cellular Automata for 2D Shape Classification

Kathryn Walker, Rasmus Berg Palm, Rodrigo Moreno, Andres Faina, Kasper Stoy, Sebastian Risi
IT University of Copenhagen
{kwal, rasmb, rodr, anfv, ksty, sebr}@itu.dk

***Abstract*— Materials with the ability to self-classify their own shape have the potential to advance a wide range of engineering applications and industries. Biological systems possess the ability not only to self-reconfigure but also to self-classify themselves to determine a general shape and function. Previous work into modular robotics systems has only enabled self-recognition and self-reconfiguration into a specific target shape, missing the inherent robustness present in nature to self-classify. In this paper we therefore take advantage of recent advances in deep learning and neural cellular automata, and present a simple modular 2D robotic system that can infer its own class of shape through the local communication of its components. Furthermore, we show that our system can be successfully transferred to hardware which thus opens opportunities for future self-classifying machines. Code available at https://github.com/kattwalker/projectcube. Video available at https://youtu.be/0TCOkE4keyc.**

I. INTRODUCTION

Many biological systems have the remarkable ability to correctly determine their anatomical structure. For example, just through the process of local communication and self-organization, cell groups can determine whether they formed a certain target shape correctly (e.g. an organ). Furthermore, they can even remodel body parts after damage. For example, the tail of a salamander can regrow and remodel into a leg after damage [7], [35], and simple organisms such as Hydra and Planaria are capable of complete morphological repair, regardless of which body part is removed [15], [36]. Their ability to self-classify general anatomy, rather than self-recognise one single target shape allows for variance between individuals, thus making the entire process more robust. For instance, the overall function and design of an organ may be the same, but the final specific shape, size or scale between different individuals of a species will vary.

Artificial engineering systems formed of thousands of individual modules with the ability to infer their own class of shape could be desirable in many applications. This ability could even enable artificial systems to go beyond nature in terms of morphological adaption to damage or to new environments by completely re-configuring their body shape. Led by the promise of versatile robotic systems, the field of modular robotics, has existed for almost 30 years. Whilst many of these systems are able to self-recognise and self-reconfigure into a specific target shape, they are missing the inherent robustness present in nature to self-classify [34]. Furthermore, much of the modular robot self-recognition work exists only in simulation and is not proven to work in hardware, limiting its usefulness for real world implementations and applications.

In this work, we build on the previous contributions from the world of modular robotics, deep learning and neural cellular automata [19], [23]. Neural cellular automata are a derivative of the simpler cellular automata and consist of a regular grid of cells where each cell can be in any one of a finite set of states. Individual cell states are then updated based on information from their neighbour's states and simple rule sets. In neural cellular automata, these simple rules are replaced by trained artificial neural networks.

We investigate how methods for shape classification through neural cellular automata [23], which have so far only been explored in perfect simulated environments, can be extended to work in hardware. Whilst the usefulness of a robust physical system is undeniable, so are the reality gap challenges of moving from simulation to hardware, particularly when dealing with multi-robot systems [6].

Therefore, the main contribution of our work is the design and implementation of a physical version of neural cellular automata capable of self-classifying 10 distinct classes of shape. Our work is able to use sparse representation to classify the overall shape of the cells, which is particularly relevant in a hardware setting where communication limitations might occur. That is, the whole "map" of the robot does not need to be communicated between every module. This way, the system also displays some inherent scale invariance when classifying differently sized shapes, even though it was not trained for it. Increasing this type of robustness is an important open challenge in modular robotics [34].

II. RELATED WORK

The work detailed here is closely related to the field of modular robotic self assembly, i.e. multiple robots that are able to communicate with one another to form an overall larger complex shape [34]. Modular robotics systems were first introduced over 30 years ago by Fukuda et al. [8], [9]. Unsurprisingly, since then, they have been researched in great detail. These modular systems hold the promise to generate versatile robots that are able to adapt to new situations, tasks and environments potentially better than their traditional counterparts.

Some attempts at designing re-configuring modular robots use centralised algorithms [14], [2], [3]. In these approaches, each module sends its own state to an external agent, which then uses all the information gathered to build a map of the current robot configuration. Clearly this method relies on the external agent for success.

Other researchers have focused on decentralised control, where each module is responsible for its own behaviour, and state changes are made based on local information gathered from neighbours and simple rules [34]. This approach is closely aligned to the concept of cellular automata in the work presented here. Usually, each module is given a specific target overall shape and its deferred local state includes some encoding of global (albeit sometimes partial) information gathered from neighbours. Note, in some cases emergent modular robot shapes have also been investigated through simple local rule interactions in decentralised agents [28].

Using a variety of different techniques and algorithms (see Thalamy et al. [34] for a comprehensive survey) this active research community has achieved excellent results. For example, a pioneering example is the work by Rubenstein et al. [26] which utilized a 1024 robot swarm [25]. In this decentralised approach, each agent was programmed with a target shape and the algorithm required to determine their necessary position within the shape. A similar swarm approach has also been carried out in 3D [32], [31] using a target CAD model, although this has only been proven to work in simulation. Other examples of work proposing different methods for modular robot configuration exist [18], [30], [20], [17] but are also carried out mainly in a purely simulated environment. The amount of systems existing in hardware is less, although some do exist for minimal amounts of modules [27], [16], [29], [38].

All the above examples (both centralised and decentralised) concentrate on algorithms to create one explicit predefined shape and are therefore not able to differentiate between different classes of shape. If, for example, the modules are required to configure into a chair, they require a specific target chair shape (e.g. a single CAD model of a chair [32]). Therefore, if the modules are unable to form that specific shape, they have failed, regardless of whether the actual shape formed meets the required specification/function. There is an inherent robustness missing in these systems; they are unable to classify the different *classes of shapes* regardless of size/scale/small changes in the overall design. As these systems are only programmed for one shape, they have no concept of their new configuration if the shape is damaged or otherwise changed through external influence. Although many of the current example would be able to recalculate their new shape, crucially they would not possess the knowledge as to whether or not this is a problem for the function of the system. They can only self-recognise, not self-classify.

One system that has used a form of modular cells for self classification is that by Randazzo et al. [23] and our work most closely aligns here. In their work, shape classification based solely on the local communication of cells (decentralised, neural cellular automata) was shown to be possible in a simulated environment [23]. The system is able to recognize MNIST digits (handwritten numbers), even with variation in the digit's shape or size. Whilst this technique, and the combination of collective intelligence with deep learning in general [24], [11], can add inherent robustness to modular robotics, it has only been shown to work in the perfect conditions of a simulated environment.

Therefore, in the work presented here, we extend the concept of robust self-classifying neural cellular automata and design a hardware tile based system. Our system is capable of self classifying 10 distinct classes of shape (i.e., the number from 0-9) and has proven robustness to changes in scale.

III. NEURAL NETWORK MODEL

Our approach is based on Neural Cellular Automata (NCA) [19], [37], [22]. A Cellular Automata (CA) is a system in which "cells" in a grid update their states according to the neighboring cells states and a set of rules. A prominent examples of a CA is Conway's game of life, which is Turing complete with binary states and just four simple rules. The NCA is a CA, in which the state of a cell is represented with a real valued vector, and the rules are functions parameterized by a neural network.

Building on previous NCA approaches [23], a single trained NCA present in each cell, uses information gathered from its neighbours to classify the overall shape based on local communication. The system runs for a set number of steps and after each step, every cell outputs a guess of what shape it believes it is a part of. In each cell the trained weights of the NCA are the same, the difference is the local information gathered from neighbours that is the input for the NCA. Using the trained NCA, after the set number of update each cell would successfully classify the overall shape. In our experiments, the cells are supposed to come to an agreement, which one of the ten different digits (Figure 1) the overall shape represents.

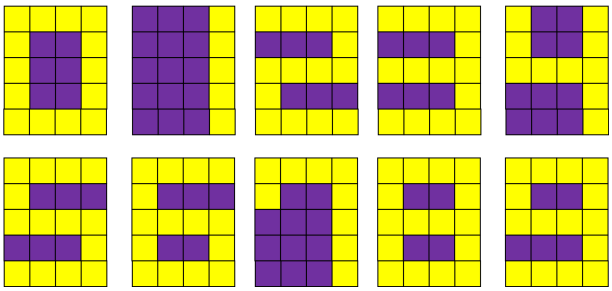


Fig. 1: The configuration of cells for each of the 10 numbers (shapes). A yellow square represents an active cells, a purple square is an empty space on the grid.

The vector valued state of cell i at step t is updated iteratively such that,

$$s_i^t = s_i^{t-1} + f_\theta(\{s_j^{t-1}\}_{j \in N(i)}), \tag{1}$$

where f_θ is a neural network computing an additive update and $N(i)$ returns the indices of the neighbors of cell i including i itself. Here we use a state vector size of 21; in preliminary experiments we saw that smaller sizes did not work as reliably. All the cell states are initialized to a vector of zeros, except the first channel which is set to one.

In the case where the cells are aligned in a grid, and the neighbors are the 3×3 surrounding cells, f_θ can be efficiently implemented using a convolutional neural network [10]. For our experiments we use a 3×3 convolution with 40 channels and a Rectified Linear Unit (ReLU) activation function [21], followed by a single 1×1 convolution also with 40 channels and relu, followed by a linear 1×1 convolution with 21 channels to match the cell state size. Note that the size of the neural network was kept conservatively small to ensure it would fit on the selected hardware micro-controller.

At any step the output of the NCA is defined as the argmax over the last C channels, where C is the amount of classes. We train the NCA to determine which one of ten different digits ($C = 10$) its shape represents (Figure 1), by minimizing the squared error between the last C channels and the one-hot encoded class label c_i,

$$\mathcal{L} = \sum_{i=1}^{N} (s_i^T[-C:] - c_i)^2 \tag{2}$$

where N is the amount of cells, $[-C:]$ indicates the last C channels and T is a random number of steps. For our experiments T is uniformly sampled between 9 and 29. It would be more natural to use the softmax cross entropy loss for classification, but we found the squared error to be more stable. The loss is minimized with stochastic gradient descent for 2500 iterations, using a batch size of 128, and the Adam optimizer with the default parameters [13]. The computations are done using Tensorflow [1].

We modify the standard NCA setup described above to work in hardware in the following ways:

Zero out kernel corners. The hardware cells can only communicate with their neighbors in the four cardinal directions and not diagonally. To simulate this we clamp the diagonal weights of the 3×3 convolutional kernels to be zero, eliminating any diagonal communication while still allowing efficient convolutional operations.

Zero out empty grid cells. The hardware cells can only communicate with other hardware cells, and not through the air. However, the convolutional neural network operates on the entire grid, and thus computes updates for the empty grid cells as well. To address this we clamp the empty grid cells to have an all zero state.

Drop updates randomly. We simulate the asynchronous nature of the hardware computation by dropping some updates at random during training. This is achieved by multiplying the computed updates by a random binary mask at each step. We randomly drop half of the updates at each step in this way.

Validate with asynchronous simulation. We validate the trained neural network in a simple asynchronous hardware simulation. At each step it samples the evaluation sequence of the N cells *with replacement*, such that 1) the evaluation order is random and 2) some cells may be evaluated more than once, and some not at all. See listing 1 for pseudo-code.

```
for i in range(n_steps):
  for j in range(len(cells)):
    random.choice(cells).evaluate()
```

Listing 1: Asynchronous hardware simulator

IV. HARDWARE DESIGN

Our hardware tiles are designed as a Printed Circuit Board (PCB) that sits on top the Arduino Mega 2560 (8 bit programmable prototyping board) as a shield. This provides a rigid body and allows us to simplify the electronics design of the module taking advantage of the 3 UART ports in the Arduino board for local communication among neighbor tiles. Table I summarises the main properties of the tile.

TABLE I: Tile properties

Property	**Value**
Base platform	Arduino Mega 2560
Dimensions	114.3x114.3 mm
Connector current rating	2 A
Input voltage	12 V
Communication	Serial UART

The shield is shaped as a square (114.3×114.3 mm) with connectors in the middle of its four sides marked as the four cardinal directions. Connectors are made of 2.54 mm pin headers and housings and are organized such that mating connectors (two male, two female) face each other when tiles are in the same orientation. The connector layout forces a specific order when connecting tiles. Nevertheless, pin headers provide a stable mechanical and electrical connection to neighboring tiles.

Each connector has 6 pins, four of which transmit power, rated for up to 2A at 12 V, and allow the tiles to be powered with only one of them connected to a power source. Tiles can still be powered independently from each other by disconnecting the Arduino power source from the shield through dedicated jumpers. The other 2 pins connect to the TTL TX and RX pins of each of the UART ports on the Arduino and are organized so that connectors match their UART pins correctly with their mating connector. Since the Arduino only has 3 hardware UART peripherals, the remaining UART connection is supplied using a software defined UART.

On the top side of the tiles, a RGB LED and a 7-segment display driven by the Arduino board, are used to indicate the number returned by the neural network. Four extra LEDs and a tactile switch can also be used to debug the tiles programming. Tiles are programmed one by one with the current version of the hardware.

V. TILE FIRMWARE

Our system is designed so that all training of the neural network is carried out offline in simulation. This setup reduces the requirements of the firmware; the weights of the trained neural network are simply stored as a fixed array in the flash memory of the micro-controller. The neural network occupies 18% of the flash memory.

As all training of the neural network is done offline, the firmware programmed onto the tiles is relatively simple,

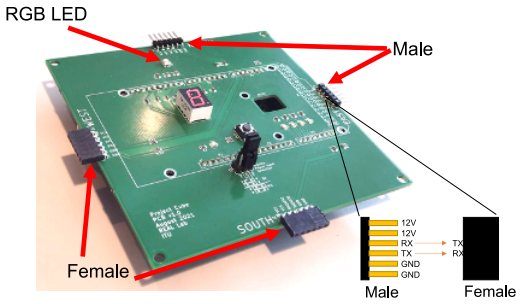


Fig. 2: Left: PCB of tile top view. Right: Connector pinout for the east side of the tile.

consisting of three main functions; *sending cell state*, *receiving neighbouring cell states* and *updating cell state*. The overall way these functions fit together is shown in Listing 2. Each cell updates after a fixed amount of time, regardless of whether it has received information from all its neighbours. For our experiments this time is set at the arbitrary value of 2 seconds. While the system can be run faster, this setting allowed us to more easily debug it.

The number of cell updates is kept at 30, which is consistent with the neural network training. The number of directions is 4 (i.e., the 4 cardinal neighbours of the tile), however this can easily be scaled to 6 neighbours if the system is expanded into 3D.

```
while(UPDATE_NUM < 30){
    for(int i = 0; i < NUM_DIRECTIONS; i++) {
            sendMessage();
        }
    for(int i = 0; i < NUM_DIRECTIONS; i++) {
            receiveMessage();
        }
    uint32_t now = millis();
    static uint32_t prev = 0;
    if((now - prev) > UPDATE_TIMEOUT_MS) {
            updateNeuralNet();
            UPDATE_NUM = UPDATE_NUM +1;
            prev = now;
          }
}
```

Listing 2: Module Firmware Overview

Sending/Receiving Data. The sending and receiving of cell states between neighbours occurs via the TTL TX and RX pins of each of the UART ports on the Arduino. Although for the majority of the program these cell states are stored as floats, during the communication phase they are converted by a linear scale to integer values between 0 - 255 to allow for communication via serial link. Although this introduces some noise to the system, it does not appear to be a problem, as suggested by the results shown below.

Updating Cell State. To update the cell state, only a neural network forward pass is required as training takes place offline. The basic method is shown in Listing 3, where n, e, s, and w are the cell states received from the north, east, south and west neighbours respectively. Note that if no neighbour is present, or the modules are out of sync and the neighbour has not yet sent a message, this data will be 0.

```
x = relu(cell_state @ perceive_kernel[1,1] +
        n @ perceive_kernel[0,1] +
        e @ perceive_kernel[1,2] +
        s @ perceive_kernel[2,1] +
        w @ perceive_kernel[1,0] +
        self.perceive_bias)

x = relu(x @ dmodel_kernel_1[0,0]
    + dmodel_bias_1)
x = x @ self.dmodel_kernel_2[0,0]
    + dmodel_bias_2
cell_state = cell_state + x
```

Listing 3: Cell Update Overview

VI. RESULTS

In this section, we show the results for three of the shapes in detail, both for the simulated version and the hardware transfer. These results are consistent with the other shapes that the neural network is trained for.

The first shape explored in detail was “4”. The hardware system is able to successfully carry out the neural cellular automata process and all the cells correctly predict that the overall shape formed is the number 4 (Figure 3a). This is indicated by both the blue LED light, and the number on the seven segment display. In Figure 3a we also show the approximate time taken for each of the updates (in this case 8 seconds before the first update, then the following updates take approx 2 seconds). This is somewhat arbitrary however, as the time between updates can be set within the firmware.

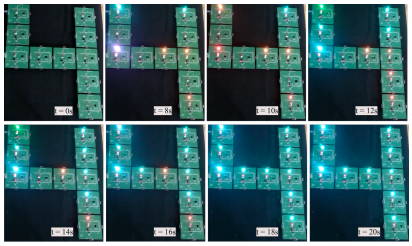
For comparison, the results from a simulated cellular automata are shown in Figure 3b. Naturally, there are differences between the individual updates. This is due to the random update nature of both the hardware and the simulated version. However, importantly, both converge on the correct solution in approximately the same number of updates (i.e. 6 in hardware, 5 in this simulated example).

Note that in the simulated version, cells that have not yet had their first update are reported as a predicted “0” value, whereas in hardware the remain unreported (i.e. no light turned on, and no value shown on the seven segment display.) Figure 3c-f show the hardware and simulation results for shapes 1 and 7.

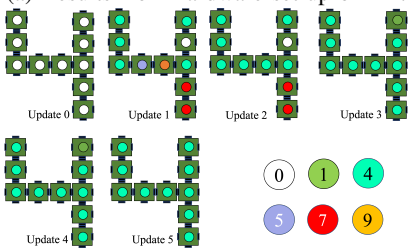
These results demonstrate that the system designed and implemented in simulation can successfully be transferred to hardware. The hardware transformation is successful for all the shapes the neural network was trained on (i.e. 0 – 9).

A. Inherent Robustness

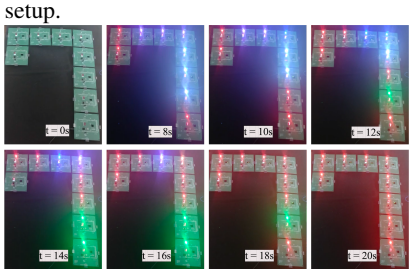
Furthermore, we carried out experiments in both simulation and hardware to demonstrate inherent robustness present as a results of this type of self-classification. In particular, we test whether the neural cellular automata is able to determine the correct class of shape (i.e. the correct number) if the overall scale of the number was changed. Note that the neural network is not trained explicitly for these different scales; it is only trained on the original 4×5 grid shapes. Successfully



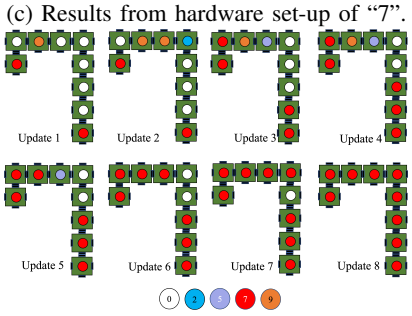
(a) Results from hardware set-up of “4”.



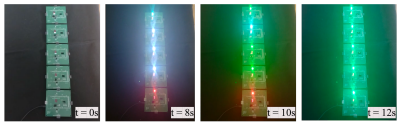
(b) Results from the simulation of the “4” setup.



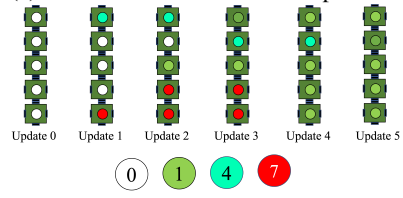
(c) Results from hardware set-up of “7”.



(d) Results from the simulation set up of “7”.



(e) Results from hardware set-up of“1”.

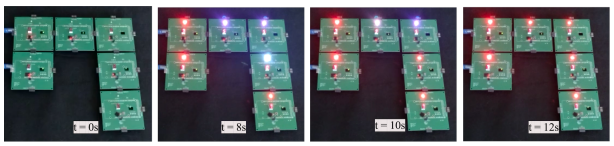


(f) Results from the simulation set up of “1”.

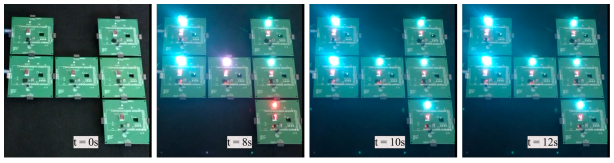
Fig. 3: Comparison of the results from simulation and hardware NCA for the numbers 4, 7, and 1.

classifying at other scales could point towards some inherent robustness in the system.

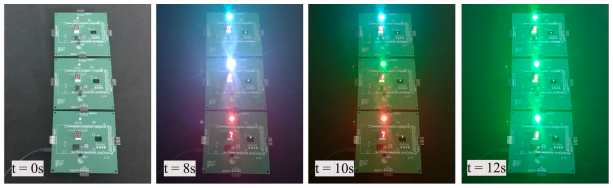
As only 20 hardware tiles are manufactured so far, we tested only scaled down robustness in hardware (i.e. can the NCA successfully classify the same shapes with fewer cells, in this case on a 3×3 grid) Furthermore, we only tested on a subset of scaled-down shapes – numbers such as 2, 3, 5, 6 and 9 would lose their overall shape when scaled to this smaller grid size. For each of these scaled down numbers/shapes the neural cellular automata was able to classify successfully (see Figure 4 for the 1, 4 and 7 examples) despite not being trained for them.



(a) Hardware of smaller version of “7”.



(b) Hardware of smaller version of “4”.



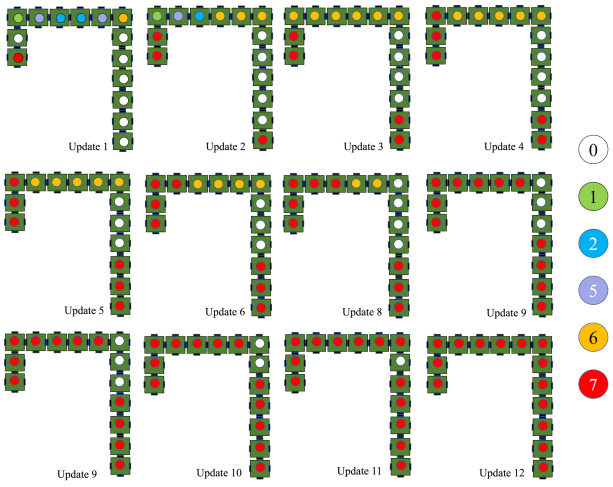
(c) Hardware of smaller version of “1”.

Fig. 4: Inherent robustness to scale changes. Shown are examples of smaller versions of the numbers, for which the neural network was not trained for, yet the cells are capable of agreeing on the correct shape.

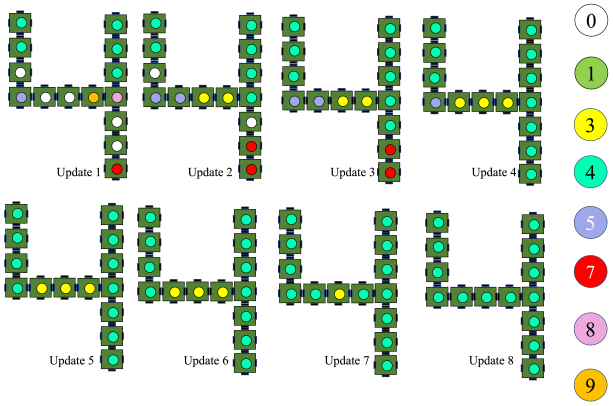
We also observed that in these scaled down scenarios, the amount of updates taken for the cellular automata to determine and agree on the shape classification was much smaller than for the original sizes. This is not unexpected as the amount of required communication between all the cells is also much less.

We also tested whether this inherent robustness was present in scaled up versions of the original numbers/shapes, this time on a 6×7 size grid and given the limited number of hardware tiles available only in simulation. All 9 scaled up versions of the shapes/numbers were once again successfully classified, with Figure 5 showing these results for 1, 4, and 7. In the future we also aim to test this scaled up version in hardware, which we believe should also work well, given the successful transfer from simulation to hardware demonstrated in the previous experiments.

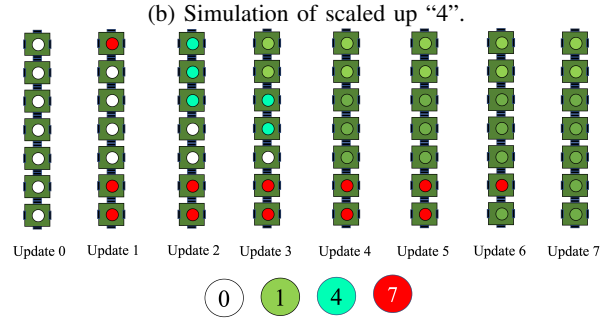
Note that in these scaled up version, the number of updates required for all cells to successfully self-classify is again larger than their original counter parts. We hypothesise that this is due to potentially important information having to



(a) Simulation of scaled up "7".



(b) Simulation of scaled up "4".



(c) Simulation of scaled up "1".

Fig. 5: Scaled up versions of the trained numbers. Note that the neural network was not trained on these larger versions but the cells are still able to agree on the correct overall shape due to the inherent robustness in the system.

travel longer distance now. For this grid size, the cellular automata is able to successfully classify every shape within the 30 update limit enforced during training. However, it is our expectation that if the grid size was increased further, this may not be the case. Therefore, if an even larger scaled up version was required, the 30 update limit during training may have to be increased.

VII. DISCUSSION AND CURRENT LIMITATIONS

The work presented here takes initial steps towards hardware systems that can determine the class of shape they belong to, only through the local communication of their components. In this section we discuss the limitations of the current research and future directions for improvements.

Firstly, the amount of tiles manufactured was minimal. Results from simulation show that the system is capable of scaling up the number of cells in each shape without loss of performance and with minimal increase in data stored on each tile (some increase in neural network size may be required). We are therefore confident that the results should also scale in hardware. Thus, one exciting area of further research is to manufacture more 2D tiles and explore the self classification of more complex shapes. Furthermore, previous work such as that by Rubenstein et al. [26] highlighted the necessity of studying natural phenomena at sufficient complexity – a strong motivation for this area of future work. We anticipate that a small redesign may be required as currently the electronics are not optimised for large power transfer through the connectors and low power consumption.

Furthermore, few machines and robots are designed and built to operate purely in 2D as this can cause problems when interacting with the real 3D world. Previously published simulated works on neural cellular automata, such as that by Sudhakaran et al. [33] (and also [12]) show success in 3D domains. In addition to scaling up the number of cells and therefore the complexity of the shapes the neural cellular automata are trained on, we also aim to investigate performance in 3D. This would be another crucial step towards self-classifying and self-modelling materials.

A current limitation with our design is that whenever a shape change occurs, all cells must be reset and re-start the classification process from the beginning. Therefore, cells cannot be added or removed *on the fly* and the system cannot adapt and update its classification automatically. One of our motivations for future research is to allow machines to detect damage and automatically adjust their behaviour to compensate. With the current firmware design this is not possible, however, on-the-fly adaption has been proved to work in simulation [23] and we expect that with a few adjustments it could also be achieved in hardware.

Finally, we hope to expand the capabilities of the current system to interact with its environment through the addition of sensors and actuators. In this context, combining the ability to self-classify with the ability to consider outcomes of multiple future actions through self-modeling [5], [4] could further increase the potential application areas of these robotic systems.

VIII. CONCLUSION

We presented a system in which modular tile-based structures can self classify their own simple shapes based on neural cellular automata and local-only communication. Due to its ability to self-classify, the system shows some inherent robustness without being trained for it, such as recognizing smaller version of the shapes it was trained

on. While currently only a proof-of-concept, the approach opens up interesting future research direction such as shape recognition of large-scale 3D structures, or self-modelling materials that can automatically detect damage.

Acknowledgements

This work was supported by a DFF-Research Project1 grant (9131- 00042B).

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] J. Baca, B. Woosley, P. Dasgupta, and C. A. Nelson. Configuration discovery of modular self-reconfigurable robots: real-time, distributed, ir+ xbee communication method. *Robotics and Autonomous Systems*, 91:284–298, 2017.

[3] D. Bie, M. A. Gutiérrez-Naranjo, J. Zhao, and Y. Zhu. An approach to the bio-inspired control of self-reconfigurable robots. In *International Conference on Bio-Inspired Computing: Theories and Applications*, pages 24–38. Springer, 2017.

[4] J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[5] B. Chen, R. Kwiatkowski, C. Vondrick, and H. Lipson. Fully body visual self-modeling of robot morphologies. *Science Robotics*, 7(68):eabn1944, 2022.

[6] M. Dorigo, G. Theraulaz, and V. Trianni. Swarm robotics: past, present, and future [point of view]. *Proceedings of the IEEE*, 109(7):1152–1165, 2021.

[7] N. Farinella-Ferruzza. The transformation of a tail into limb after xenoplastic transplantation. *Experientia*, 12(8):304–305, 1956.

[8] T. Fukuda and Y. Kawauchi. Cellular robotic system (cebot) as one of the realization of self-organizing intelligent universal manipulator. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 662–667. IEEE, 1990.

[9] T. Fukuda and T. Ueyama. *Cellular robotics and micro robotic systems*, volume 10. World Scientific, 1994.

[10] W. Gilpin. Cellular automata as convolutional neural networks. *Physical Review E*, 100(3):032402, 2019.

[11] D. Ha and Y. Tang. Collective intelligence for deep learning: A survey of recent developments. *arXiv preprint arXiv:2111.14377*, 2021.

[12] K. Horibe, K. Walker, and S. Risi. Regenerating soft robots through neural cellular automata. In *EuroGP*, pages 36–50, 2021.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] K. D. Kotay and D. L. Rus. Algorithms for self-reconfiguring molecule motion planning. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, volume 3, pages 2184–2193. IEEE, 2000.

[15] M. Levin, J. Selberg, and M. Rolandi. Endogenous bioelectrics in development, cancer, and regeneration: drugs and bioelectronic devices as electroceuticals for regenerative medicine. *Iscience*, 22:519–533, 2019.

[16] C. Liu and M. Yim. Configuration recognition with distributed information for modular robots. In *Robotics Research*, pages 967–983. Springer, 2020.

[17] N. Mathews, A. L. Christensen, R. O'Grady, F. Mondada, and M. Dorigo. Mergeable nervous systems for robots. *Nature communications*, 8(1):1–7, 2017.

[18] Y. Meng, Y. Zhang, and Y. Jin. Autonomous self-reconfiguration of modular robots by evolving a hierarchical mechanochemical model. *IEEE Computational Intelligence Magazine*, 6(1):43–54, 2011.

[19] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin. Growing neural cellular automata. *Distill*, 2020. https://distill.pub/2020/growing-ca.

[20] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 441–448. IEEE, 1994.

[21] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[22] S. Nichele, M. B. Ose, S. Risi, and G. Tufte. Ca-neat: evolved compositional pattern producing networks for cellular automata morphogenesis and replication. *IEEE Transactions on Cognitive and Developmental Systems*, 10(3):687–700, 2017.

[23] E. Randazzo, A. Mordvintsev, E. Niklasson, M. Levin, and S. Greydanus. Self-classifying mnist digits. *Distill*, 5(8):e00027–002, 2020.

[24] S. Risi. The future of artificial intelligence is self-organizing and self-assembling. *sebastianrisi.com*, 2021.

[25] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE international conference on robotics and automation*, pages 3293–3298. IEEE, 2012.

[26] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[27] B. Salemi, P. Will, and W.-M. Shen. Autonomous discovery and functional response to topology change in self-reconfigurable robots. In *Complex Engineered Systems*, pages 364–384. Springer, 2006.

[28] I. Slavkov, D. Carrillo-Zapata, N. Carranza, X. Diego, F. Jansson, J. Kaandorp, S. Hauert, and J. Sharpe. Morphogenesis in robot swarms. *Science Robotics*, 3(25):eaau9178, 2018.

[29] A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi, and A. J. Ijspeert. Roombots: A hardware perspective on 3d self-reconfiguration and locomotion with a homogeneous modular robot. *Robotics and Autonomous Systems*, 62(7):1016–1033, 2014.

[30] A. Spröwitz, S. Pouya, S. Bonardi, J. Van Den Kieboom, R. Möckel, A. Billard, P. Dillenbourg, and A. J. Ijspeert. Roombots: reconfigurable robots for adaptive furniture. *IEEE Computational Intelligence Magazine*, 5(3):20–32, 2010.

[31] K. Stoy. Using cellular automata and gradients to control self-reconfiguration. *Robotics and Autonomous Systems*, 54(2):135–141, 2006.

[32] K. Stoy and R. Nagpal. Self-reconfiguration using directed growth. In *In Proc. 7th Int. Symp. on Distributed Autonomous Robotic Systems*. Citeseer, 2004.

[33] S. Sudhakaran, D. Grbic, S. Li, A. Katona, E. Najarro, C. Glanois, and S. Risi. Growing 3d artefacts and functional machines with neural cellular automata. *arXiv preprint arXiv:2103.08737*, 2021.

[34] P. Thalamy, B. Piranda, and J. Bourgeois. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems*, 120:103242, 2019.

[35] W. A. Vieira, K. M. Wells, and C. D. McCusker. Advancements to the axolotl model for regeneration and aging. *Gerontology*, 66(3):212–222, 2020.

[36] M. C. Vogg, B. Galliot, and C. D. Tsiairis. Model systems for regeneration: Hydra. *Development*, 146(21):dev177212, 2019.

[37] N. Wulff and J. A. Hertz. Learning cellular automaton dynamics with neural networks. *Advances in Neural Information Processing Systems*, 5:631–638, 1992.

[38] M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3d metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.