Module Addition-Aware Dynamic Shape Recognition for Lattice-Based Modular Robots

1st Jad Bassil
Univ. Franche-Comté
FEMTO-ST Institute, CNRS
1 cours Leprince-Ringuet, 25200, Montbéliard, France.
jad.bassil@univ-fcomte.fr

2nd Benoît Piranda
Univ. Franche-Comté ,
FEMTO-ST Institute, CNRS
1 cours Leprince-Ringuet, 25200, Montbéliard, France.
benoit.piranda@univ-fcomte.fr

3rd Abdallah Makhoul
Univ. Franche-Comté ,
FEMTO-ST Institute, CNRS
1 cours Leprince-Ringuet, 25200, Montbéliard, France.
abdallah.makhoul@univ-fcomte.fr

4th Julien Bourgeois
Univ. Franche-Comté ,
FEMTO-ST Institute, CNRS
1 cours Leprince-Ringuet, 25200, Montbéliard, France.
julien.bourgeois@univ-fcomte.fr

***Abstract*—Self-Reconfigurable Modular Robots typically consist of high number of modules with uniform docking interfaces, allowing them to transform into various shape. Recognizing the shape of such a system composed of hundreds of modules is a significant challenge. Given a new configuration, a modular robots system must be able to determine and update its shape dynamically and in a distributed manner. In a previous work, we developed an algorithm that identifies overlapping boxes to cover the entire robot configuration through message-passing, enabling robots to determine a representation of their current shape. However, this algorithm was static and did not react to changes in real time. In this paper, we introduce an updated shape recognition algorithm that dynamically and in real-time recognizes the addition of modules to update the shape description of the entire configuration using local information. The dynamic algorithm to update the shape description is tested in a simulated environment and compared to re-executing the shape recognition algorithm on the whole configuration. The results show the efficiency of our algorithm in updating the robot's current shape in real time.**

***Index Terms*—Shape recognition, modular robots, distributed algorithm.**

I. Introduction

The project we are interested in concerns the creation of a Programmable Matter associated with a digital twin. The aim of this project is to speed up and make more efficient objects design processes in industry, by using Programmable Matter. The illustration presented Fig. 1 presents a system where spherical tiny robots are forming the material of the small object on the bottom left of the picture and communicate the global shape they are defining to a computer-aided design system.

Programmable Matter is a substance made up of a very large number of small robotic modules that fit together to form an object. As these modules are programmable and equipped with sensors and actuators, they bring new capabilities to this material, such as the ability to change color or shape on demand, or to detect contact or deformation with an external body.

979-8-3315-1995-7/25/$31.00 ©2025 IEEE

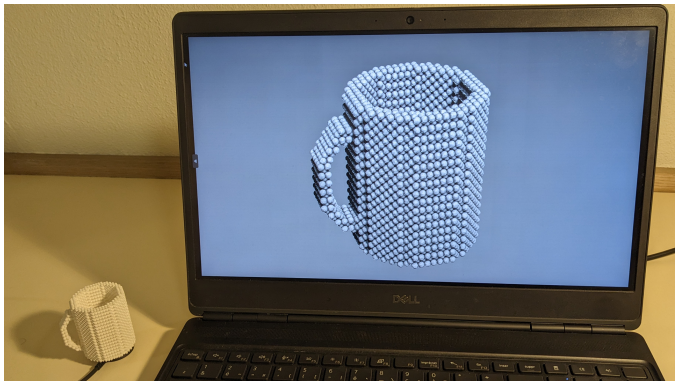


Fig. 1: Artist view of the goal of the project.

The notion of a digital twin is important for interacting with this material. The aim of this project is to have a set of robots that communicates with a CAD system to synchronize modifications to the digital model with transformations made to the real object, and vice-versa. If the real object is deformed, the same deformation is applied to the digital model, and if an operator transforms the digital model, the real object made of robotic modules reconfigures itself to achieve the same shape.

In the context of a self-reconfiguration algorithm, it may be important that all the modules know the current configuration by keeping an updated description in their memory. Global knowledge of the shape enables more efficient self-reconfiguration, for example by computing the difference between the current configuration and the final configuration, modules can make informed distributed decisions to plan the self-reconfiguration towards the final configuration.

We consider modules placed in a 3D regular lattice, allowing them to move between two cells as long as their docking

positions remain within the grid. The modules can also communicate by exchanging messages with their directly attached neighbors via latching interfaces.

A distributed message-passing shape recognition algorithm is proposed in [1]. In this distributed algorithm modules exchange messages to find a set of overlapping boxes that covers the entire configuration. The union of these boxes forms a description of the configuration's shape.

In this paper, we propose a dynamic version of the global solution presented in [1], allowing to efficiently update the set of the pre-calculated boxes when a new module is added to the configuration or when an exiting module changes position. We show that these update operations are much faster than the complete recalculation of a new set of boxes because they only affect an area close to the added module.

The paper is structured as follows. Section II surveys the related work. Section III covers preliminaries, including a concise description of the base shape recognition algorithm and the assumptions of the underlying system. Section IV details the simulation experiments and compares the proposed dynamic method that consists of local updates against the re-execution of the shape recognition algorithm on the entire configuration. Finally, Section V concludes the paper and suggests potential future research directions.

II. Related Work

Some of the technological challenges involved in achieving this goal have been widely discussed in the literature. First of all the creation of such autonomous and self-reconfigurable robotic modules. A complete tiny autonomous robot with reconfigurable capabilities is not available yet, but we can mention the works that propose several shapes of robots associated with locomotion methods.

One of the first solution was proposed by Rus & Marsette in [2] presenting the Crystalline robot system A set of modules are aggregated together to form a more complex distributed robot system. The movement of the robots is based on expanding and contracting a pair of neighbor units. These local motions produce the self-reconfiguration of the global system [3]. Another well-known example is presented by Romanishin et al. in [4], where the self-transforming M-Blocks (2.0) robots can jump, spin, flip, and identify each other. M-Blocks offer many crucial features for making programmable matter but are still very big and heavy to be elements of an object combining thousands of them. RoomBots proposed by Sproewitz and al [5] are cubic robots whose two moving parts can rotate relatively to each other to move one or more robots in a grid. This solution offers slower but more precise and guaranteed movement than M-blocks, but the size of the $12\,\mathrm{cm}$ is still too large for our application. Pebbles, proposed by Gilpin et al. [6] are much smaller modular robots, these $1\,\mathrm{cm}$ cubic modules can detach from their neighbors on demand, representing a partial solution to our need.

The Femto-st Institute's Programmable Matter project aims to create quasi-spherical robots called *3D Catom* [7], [8] that can attach to up to 11 neighbors in a lattice and move over each other by rotation. This highly ambitious project is currently capable of attaching and detaching modules measuring $3.9\,\mathrm{mm}$ in diameter. This solution will, in time, enable the production of self-reconfigurable Programmable Matter.

The algorithmic problem of planning the self-reconfiguration of modules is addressed in numerous articles. Kawano proposes many algorithms [9]–[11] considering tunneling capabilities of modules (the capability of sliding cubic shape robots inside a large set). With the same hypothesis, but using spherical robots, Lengiewicz et al. [12] propose a distributed auto-reconfiguration method allowing the parallel movement of several modules at the risk of collisions, using a distributed version of the MaxFlow algorithm. In [13] Bassil et al. propose a different solution for the distributed auto-reconfiguration algorithm using the movement of modules inside a porous structure, this method coupled with a scaffold overlay method, reduces the number of modules to define an object and increase modules movements flow and parallelism.

Finally, the problem that interests us here is the ability of a set of modules to self-diagnose and return to a master machine the shape constructed by a large set of elements. The work presented in [1] proposes an algorithmic solution that allows the set of modules to describe the shape as a set of overlapping boxes whose union exactly covers the shape of the set of modules.

Considering the programmable matter context with very large sets of modules, dimensions along axis are wide. For example, for the small cup presented in Fig. 1, the dimensions of the lattice are $40 \times 50 \times 40$ and it needs 20 715 *3D Catoms*. We need local algorithms to optimize treatments like adding or removing modules on an existing object without recomputing processes over the complete set.

III. Algorithm Description

A. Preliminary

The distributed algorithm presented in [1] constructs a set of rectangular boxes aligned with the axes of the maximum-size grid. Figure 2 shows how it works. It proceeds in three stages: first, modules that have no neighbor in the "back" direction (drawn in red) build lines along the "front" axis. Then, these lines are transformed into rectangles using an algorithm that makes them grow along the "right" axis from their bottom left corner (in color in b. section of the figure). Finally, the 2D rectangles are extended upward to form 3D boxes with no empty positions.

The efficiency of this algorithm is due to its parallelization. For example, in the first step, all segments oriented "back $\rightarrow$ front" are produced simultaneously, giving a complexity of $O(Y)$ where Y is the size of the configuration along the $\overrightarrow{y}$ axis. Finally the proposed algorithm is in $O(X+Y+Z)$ where X, Y and Z are respectively the size of the bounding box of the configuration.

The algorithm combines message exchanges and short processing operations at module level to determine which message to send when a message is received. These processes boil down

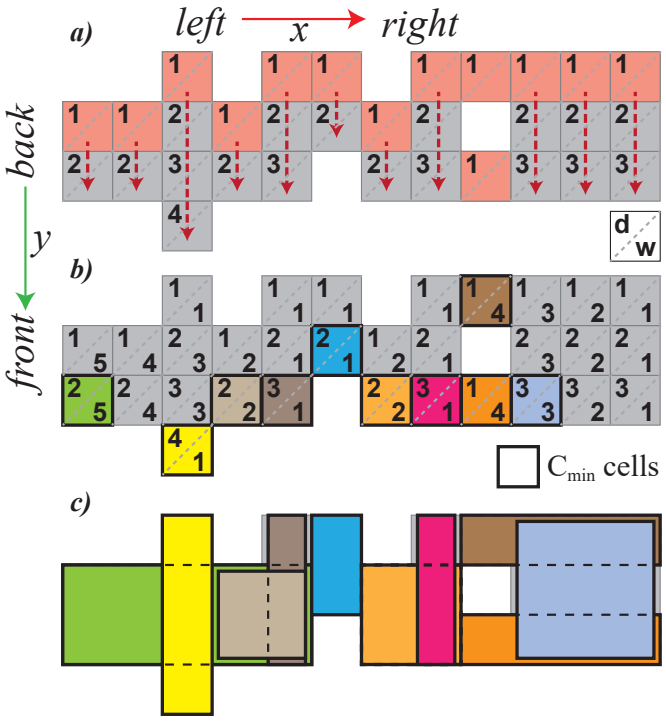


Fig. 2: Distributed computation of the boxes on 2D. a) Computation of vertical distance d. b) Election of C_{min} cells and computation of w. c) Boxes covering the plane using C_{min} cells colors.

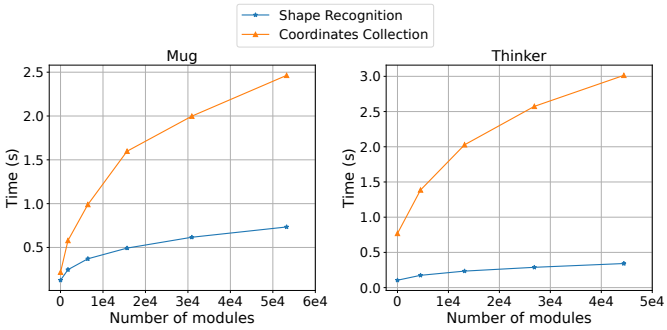


Fig. 3: The difference in time between the shape recognition algorithm proposed in [1] and an exhaustive coordinates collection while varying the size of the two configurations shown in Fig. 5.

to a few memory accesses and conditional operations. We therefore consider message processing time to be negligible compared with communication time. In the case of *Blinky Block*, in [1] the authors propose a numerical model for estimating the transfer time of a message as a function of its size. Fig. 3 shows that the shape recognition algorithm is faster and scales better than an exhaustive coordinates collection.

We assume that all modules share the same 3D coordinate system and store their coordinates in memory. This can be done efficiently in lattice-based modular robots, as explained in [14], [15]. we use the axis orientation and direction notation shown Fig. 2. A Box B is defined by two vectors $C_{min}(x_{min}, y_{min}, z_{min})$ and $C_{max}(x_{max}, y_{max}, z_{max})$ such that:

$$X(x,y,z) \in B \Leftrightarrow \begin{cases} x_{min} \leq x \leq x_{max} \\ y_{min} \leq y \leq y_{max} \\ z_{min} \leq z \leq z_{max} \end{cases} \quad (1)$$

Modules can only communicate with their directly attached neighbors through their latching interfaces. They can detect if an interface is connected to a neighboring module, indicating the presence of a neighbor in that direction.

B. Module addition dynamic update algorithm

This section presents a dynamic algorithm to efficiently update the set of boxes with a module addition. This is achieved by performing updates locally using local information about the boxes surrounding the newly added module, thereby avoiding a complete rerun of the shape recognition algorithm described in Section III-A on the entire configuration. This approach saves communication resources and reduces the update time and the number of communications.

When a module M is added, it performs the following steps, which are illustrated in Fig. 4:

1) **Initialization**: Upon the addition of a new module M with coordinates (x_M, y_M, z_M), a new box $B_{new} = ((x_M, y_M - k, z_M), (x_M, y_M + l, z_M))$ is created. This box spans vertically from $y_M + l$ to $y_M - k$, where l and k represent the number of connected modules along the positive and negative directions of the Y-axis respectively. This will create a box formed by a connected line of modules.
2) **Neighbor boxes adjustment** (cf. Algo. 1, lines 8-12): The added module M retrieves neighboring boxes N_{boxes} surrounding it by querying adjacent modules that will send it the set of boxes they belong to. Then, for each box $B_i \in N_{boxes}$, the dimensions of B_i are updated so they become aligned with B_{new}.
3) **Boxes combination** (cf. Algo. 1 lines 13-19): The newly added box, B_{new} can be combined with other boxes in N_{boxes} into a larger box which can results in reducing the number of boxes after addition. The algorithm determines if B_{new} can be combined with any $B_i \in N_{boxes}$ by verifying whether their intersection results in an empty set when subtracted from their combined bounding box. Specifically, if $\text{BoundingBox}(B_i, B_{\text{new}}) - (B_i \cap B_{new}) = \emptyset$, this condition verifies that there are no empty positions within the combined bounding box of B_i and B_{new}. Consequently, B_{new} is updated to encompass the bounding box of B_i and B_{new}. This process continues in an iterative manner until no more combining of boxes in N_{boxes} with B_{new} is possible.
4) **Boxes update** (cf. Algo. 1 lines 20-22) : The combining process initiated by the newly added module M results in a new box B_{new} that will likely include a set of modules that are different from M. Once B_{new} is determined, module M broadcast a message

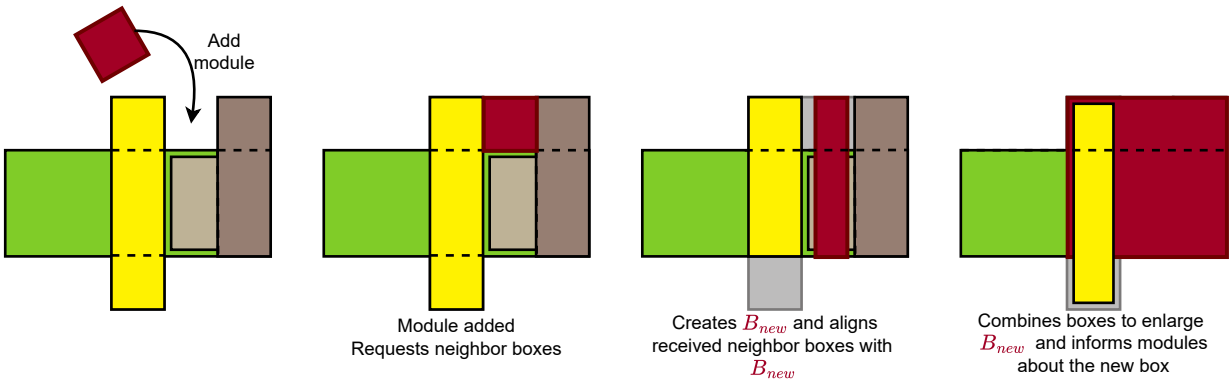
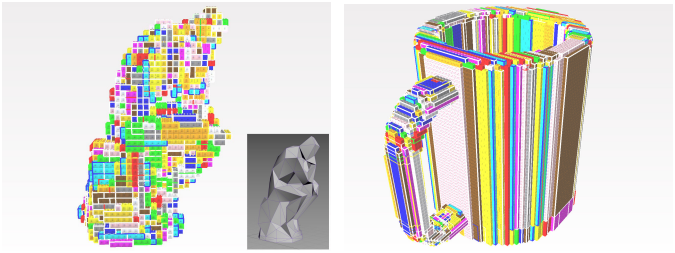


Fig. 4: Illustration of the sequence of actions performed by the module highlighted in red after its addition to update the set of boxes.



(a) (b)

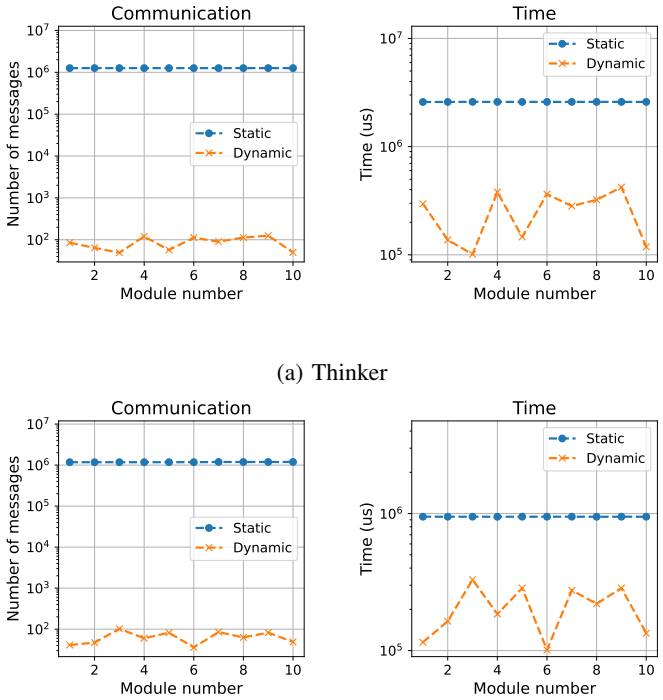
Fig. 5: (a) "Thinker" configuration with 45 720 *Blinky Blocks* and (b) "Mug" configuration with 53 484 *Blinky Blocks* captured from *VisibleSim*.

containing B_{new} to all modules within B_{new}. Upon receiving this message, the modules will update the set of boxes they belong to by adding B_{new} and removing the boxes whose dimensions are fully included in B_{new}. Furthermore, module M also reports B_{new} by sending a message to the connected computer to inform it of the updates on the configuration in real time.

IV. Experiments

We implemented the algorithm in *VisibleSim* [16], a discrete event-based simulator for distributed modular robotic systems that supports *Blinky Blocks* [17]. The *Blinky Blocks* system consists of centimeter-sized blocks connected to each other using magnets in a square cubic lattice configuration as shown in 8. Each block is approximately a 40 millimeter cube equipped with processing, storage, and communication features. Communication between *Blinky Blocks* occurs through serial links with directly connected neighbors, utilizing packets with a payload size of 227 B.

In order to assess the proposed dynamic shape recognition approach for module addition, we compare the communications and time needed to update the set of boxes between the static method, which involves re-executing the shape recognition on the whole configuration, and the dynamic method



(a) Thinker

(b) Mug

Fig. 6: Results for messages and time for both "Thinker" and "Mug" shapes.

suggested in this paper, which updates the set of boxes locally upon a module addition.

We have done the comparison on two different configurations shown in Fig. 5:

1) Thinker: The thinker statue is defined by a low-resolution mesh presenting irregularities on its border and initially requires 1 864 boxes to cover the entire configuration.
2) Mug: A mug shape composed of 53 484 *Blinky Blocks* with few irregularities initially requires 305 boxes to

Algorithm 1: Dynamic algorithm for updating box set upon module addition

Event: New module M with coordinates (x_M, y_M, z_M) is added
Output: Updated set of boxes

// Step 1: Create a new box
$l \leftarrow$ Number of connected modules in the back direction of M
$k \leftarrow$ Number of connected modules in the front direction of M
$y_{\text{max}} \leftarrow y_M + l$
$y_{\text{min}} \leftarrow y_M - k$
$B_{\text{new}} \leftarrow (\{x_M, y_{\text{max}}, z_M\}, \{x_M, y_{\text{min}}, z_M\})$

// Step 2: Neighbor boxes adjustment
$N_{\text{boxes}} \leftarrow M$.requestNeighborBoxes()
foreach $B_i \in N_{boxes}$ **do**
  for $k \leftarrow 0$ **to** 2 **do**
    if $B_i.C_{min}[k] \leq B_{new}.C_{min}[k]$ ***and*** $B_i.C_{max}[k] > B_{new}.C_{max}[k]$ **then**
      $B_i.C_{\text{max}}[k] \leftarrow B_{\text{new}}.C_{\text{max}}[k]$
    else if $B_i.C_{max}[k] \geq B_{new}.C_{max}[k]$ ***and*** $B_i.C_{min}[k] < B_{new}.C_{min}[k]$ **then**
      $B_i.C_{\text{min}}[k] \leftarrow B_{\text{new}}.C_{\text{min}}[k]$

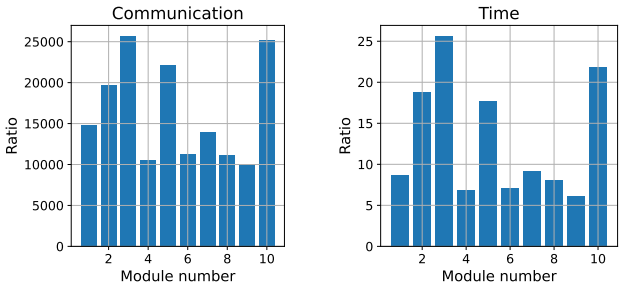
// Step 3: Combine boxes
$combined \leftarrow true$
while *combined* **do**
  foreach $B_i \in N_{boxes}$ **do**
    if $BoundingBox(B_i, B_{new}) - intersection(B_i, B_{new}) = \emptyset$ **then**
      $B_{\text{new}} \leftarrow \text{BoundingBox}(B_i, B_{\text{new}})$
    else
      $combined \leftarrow false$

// Step 4: Inform modules in new box
foreach *module* m *in* B_{new} **do**
  Inform m of B_{new} to update the set of boxes to which it belongs.

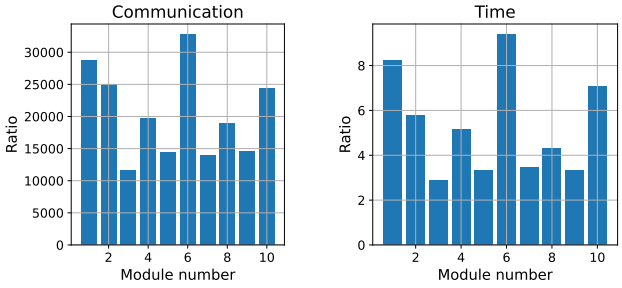
// Step 5: Report the new box
send B_{new} to the connected computer

cover the entire configuration.

For each of the configurations, 10 modules are sequentially added to the configurations in random positions. Upon each addition, the dynamic and static shape recognition algorithms are executed and compared. Fig. 6 displays the difference in time and communication and Fig. 7 illustrates the difference ratios between the static and dynamic methods. It can be seen that the dynamic method is faster and requires less time and communications to update the set of boxes in both configurations. On average, the dynamic method is 10,08 times faster and requires 14 557 times fewer message exchanges on the



(a) Thinker



(b) Mug

Fig. 7: Ratio of communication and time between static and dynamic method for both "Thinker" and "Mug" shapes.

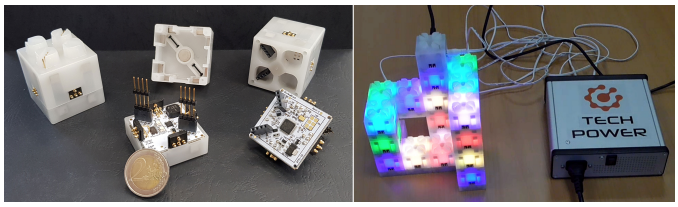


Fig. 8: *Blinky Blocks* v2 modular robotic system.

Thinker configuration and it is 4,52 times faster and requires 18 283 times fewer messages on the Mug configuration.

The fluctuations in time and number of communications when using the dynamic method, as shown in Fig. 6 and Fig. 7, are primarily due to the position of the added module. If the added module results in the formation of a new large box, more messages are needed to inform the modules within that box about its presence. Additionally, if the added module is distant from the connected computer, more messages are required to reach it.

V. Conclusion and Future Works

In this paper, a dynamic algorithm to update the shape description of a lattice-based modular robot configuration upon a module addition is proposed. The proposed method can be applied to the shape recognition algorithm based on overlapping boxes. It aims to update the set of boxes based on local information when a new module is added.

We evaluated the proposed dynamic algorithm in simulation on *Blinky Blocks* modules and compared the communications and time needed to update the set of boxes with re-executing the initial shape recognition algorithm on two different configurations. The results demonstrated that the dynamic method yielded a tenfold improvement in processing time and a ten-thousandfold reduction in communication requirements to update the configuration description.

Our future objective is to design a dynamic algorithm capable of addressing module removal and configuration splitting, which pose additional challenges compared to module addition, such as the potential disconnection of configurations and the disruption of the communication graph.

REFERENCES

[1] J. Bassil, J.-P. A. Yaacoub, B. Piranda, A. Makhoul, and J. Bourgeois, "Distributed shape recognition algorithm for lattice-based modular robots," in *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, 2023, pp. 85–91.

[2] D. Rus and M. Vona, "A physical implementation of the self-reconfiguring crystalline robot," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 1726–1733.

[3] ——, "Crystalline robots: Self-reconfiguration with compressible unit modules," *Autonomous Robots*, vol. 10, no. 1, pp. 107–124, 2001.

[4] J. W. Romanishin, K. Gilpin, S. Claici, and D. Rus, "3d m-blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 1925–1932.

[5] A. Sproewitz, P. Laprade, S. Bonardi, M. Mayer, R. Moeckel, P.-A. Mudry, and A. J. Ijspeert, "Roombots—towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 1126–1132.

[6] K. Gilpin, A. Knaian, and D. Rus, "Robot pebbles: One centimeter modules for programmable matter through self-disassembly," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 2485–2492.

[7] B. Piranda and J. Bourgeois, "Designing a quasi-spherical module for a huge modular robot to create programmable matter," *Autonomous Robots*, vol. 42, no. 8, pp. 1619 – 1633, dec 2018. [Online]. Available: https://publiweb.femto-st.fr/tntnet/entries/14451/documents/author/data

[8] Y. Peng, G. Carichner, Y. Kim, L.-Y. Chen, R. Tribhout, B. Piranda, J. Bourgeois, D. Blaauw, and D. Sylvester, "A high-voltage generator and multiplexer for electrostatic actuation in programmable matter," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 4, pp. 915–928, 2023.

[9] H. Kawano, "Complete reconfiguration algorithm for sliding cube-shaped modular robots with only sliding motion primitive," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 3276–3283.

[10] ——, "Full-resolution reconfiguration planning for heterogeneous cube-shaped modular robots with only sliding motion primitive," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 5222–5229.

[11] ——, "Distributed tunneling reconfiguration of sliding cubic modular robots in severe space requirements," in *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 1–15.

[12] J. Lengiewicz and P. Holobut, "Efficient collective shape shifting and locomotion of massively-modular robotic structures," *Auton. Robots*, vol. 43, no. 1, pp. 97–122, 2019. [Online]. Available: https://doi.org/10.1007/s10514-018-9709-6

[13] J. Bassil, B. Piranda, A. Makhoul, and J. Bourgeois, "Repost: Distributed self-reconfiguration algorithm for modular robots based on porous structure," in *IEEE RSJ International Conference on Intelligent Robots and Systems (IROS 2022)*, Kyoto, Japan, oct 2022.

[14] B. Piranda, F. Lassabe, and J. Bourgeois, "Disco: A multiagent 3d coordinate system for lattice based modular self-reconfigurable robots," 2023.

[15] P. Hołobut, P. Chodkiewicz, A. Macios, and J. Lengiewicz, "Internal localization algorithm based on relative positions for cubic-lattice modular-robotic ensembles," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3056–3062.

[16] P. Thalamy, B. Piranda, A. Naz, and J. Bourgeois, "Visiblesim: A behavioral simulation framework for lattice modular robots," *Robotics and Autonomous Systems*, p. 103913, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889021001986

[17] "Blinky blocks: Programmable matter," https://www.programmable-matter.com/technology/blinky-blocks.