Distributed Shape Recognition Algorithm for Lattice-Based Modular Robots

Jad Bassil[†], Jean-Paul A. Yaacoub[†], Benoît Piranda[†], Abdallah Makhoul[†] and Julien Bourgeois[†]

***Abstract*— Lattice-based modular robots are composed of modules arranged on a lattice and forming 3D shapes. This paper presents a fully distributed algorithm executed by each module of a large lattice-based modular robot to determine their current shape using a small memory footprint and fast convergence. The algorithm consists of finding overlapping boxes that cover the entire robot configuration using message-passing. Thus, allowing robots to determine a representation of their current shape. The discovery of a distributed representation of the current shape of modular robots provides valuable information that can be either used to facilitate the distributed self-reconfiguration planning or to efficiently send the current shape to a connected computer. The algorithm is executed in a simulated environment and compared with a classic coordinates collection method in order for modules to send their current global shape to a computer to be used by an interactive CAD software. The obtained results show the efficiency of our algorithm in detecting the current shape of the robot, while also outperforming the existing coordinate collection algorithm.**

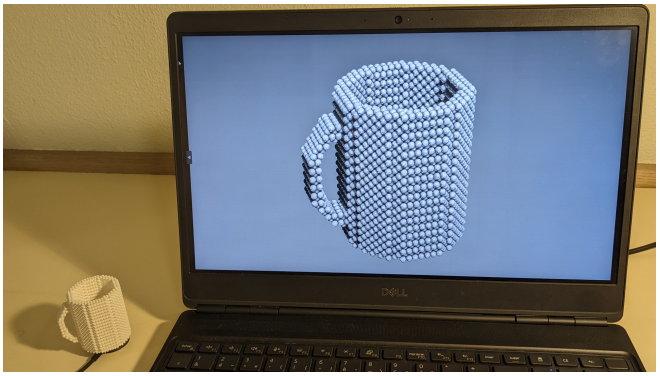


Fig. 1: Illustration of the objective made of a system where spherical robots forming the material of the small object on the left communicate the global shape they are defining to a computer-aided design system.

I. Introduction

The concept of programmable matter refers to materials that can change their physical properties in response to external stimuli or user input. The Programmable Matter will be a new tool to design objects since it offers reusable, dynamic, interactive and intelligent interfaces. The idea is to connect the matter in real time with a computer running a Computer-Aided Design (CAD) software. Using the CAD software, a shape is described and then transmitted to the matter. On reception, the matter will reconfigure itself into the given shape, that is self-reconfiguration, or by responding to physical user interaction with the matter. These evaluations in the physical matter are reflected by the digital object in the CAD software, creating an interactive design tool [1] as shown in Fig. 1.

Self-reconfigurable modular robots, which are made up of individual modules that can be reconfigured to form different shapes and structures, are a prime example of programmable matter. They are made up of individual modules with computational and communication capabilities. They can be attached, detached, and moved to change the overall shape. They offer the required properties to achieve such a matter which includes: programmability, reconfigurability, and interactivity. Future hardware like *3D Catoms* [2], which are millimeter-scale quasi-spherical modules will produce very precise matter, but currently it is already possible to assemble cubic robots using *Blinky Blocks* [3], in order to create a shape connected to a computer.

One of the challenges in achieving such a matter is shape representation. In [4], Tucci et al. show how to encode the representation of a shape of an object in a compact memory efficient way using a Constructive-Solid Geometry (CSG) model, usable to transfer a shape from the CAD to the set of robots which gives them enough information to self-reconfigure. Nevertheless, it is also essential to allow the modules to discover a representation of their current configuration, which is useful for sending it back to the connected computer to update the digital object.

Moreover, in a large modular robot, modules have limited local knowledge of the entire configuration. They can only access their directly connected neighbors. Thus, allowing modules to recognize the whole shape of their configuration can facilitate distributed self-reconfiguration planning, which consists of finding the sequence of movements to reconfigure into a goal shape. Knowing the current configuration modules can calculate the difference between the current and the goal shape to optimize the movements and the rearrangements to be made to reach the goal shape while ensuring safe and mechanically stable movements.

In this paper, we propose a distributed algorithm that allows a set of robots to describe the current shape that it is forming. Our algorithm defines a set of overlapping boxes to cover the entire robot configuration, allowing for the determination of a representation of its current shape. The unique constraint of a box is that it covers a complete set of modules without any holes. By using a distributed

[†]All authors are with Univ. Franche-Comté, FEMTO-ST Institute, CNRS, 1 cours Leprince-Ringuet, 25200, Montbéliard, France. {first}.{last}@femto-st.fr

approach, each module of the robot can communicate with its directly attached neighbors to collectively determine the global robot's shape.

In Section II we present the related works. In Section III we present the distributed algorithm to detect the overlapping boxes. Then, we analyze and express its time and communication complexity in Section IV. In Section V we present the experiments carried out, which compare the shape recognition method with a coordinates collection method to inform the current configuration to a computer. The paper is concluded and future works are mentioned in section VI.

II. Related Works

The configuration recognition problem, which consists of matching and mapping a configuration to a library of configurations, is studied in [5], [6], [7]. First, a discovery phase is executed to find a representation of the current configuration as an inter-connectivity graph, where nodes represent modules and edges represent the connections between the modules. Then, they match the graph with an existing one and map the physical modules to their logical one.

In [8], the authors solve the matching problem with a distributed goal recognition algorithm that verifies if a configuration matches a given goal shape without the need to discover the whole configuration. In [5] a distributed real-time algorithm is presented for configuration discovery. It allows modules to discover other modules using wireless infrared communication and construct the connectivity graph.

Connectivity graphs suffer from scalability issues since they depend on the number of modules which might increase drastically, especially when building a high fidelity programmable matter with millimeter scale robots. Furthermore, in lattice-based modular robots, we can exploit geometric information to create a compact shape representation.

In [9], [10] the authors propose to transform a CAD model into overlapping bricks to make it easier for the modules to identify their position relative to goal configuration which is required for the self-reconfiguration process. In [4] the authors proposed to use a CSG tree. In this model coming from the image synthesis domain, the leaves of the tree contain basic geometrical objects and the intern nodes contain geometrical transformations and combination operators (union, intersection, or difference) to form the final shape on the root. The objective of these methods is to encode a shape using a centralized computer to transmit it to the module to self-reconfigure into it.

In this paper, our aim is to propose a distributed algorithm that allows modules forming a large lattice-based modular robot to discover the overall shape of their configuration using overlapping boxes whose union forms the configuration using neighbor-to-neighbor message passing.

III. Algorithm Description

The main idea of the algorithm is to find a set of full boxes that cover the whole configuration. In this section, we describe the distributed algorithm, shown in Algorithms 1 and 2 for finding the boxes on modular robotic systems where modules communicate using message-passing with their directly attached neighbors.

We assume that all modules share the same 3D coordinate system and each module stores its coordinates in its memory. This can be done efficiently in lattice-based modular robots, as explained in [11], [12]. The algorithm is not affected by the orientation of the coordinate axis. However, for simplicity, we use the orientation of the axis and the direction notation as shown in Fig. 2. A Box B is defined by two vectors $C_{min}(x_{min}, y_{min}, z_{min})$ and $C_{max}(x_{max}, y_{max}, z_{max})$.

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

Let M denote the set of modules. A straight sequence of connected modules in one direction is referred to as a line of modules. The initial step of the algorithm involves determining the values d_m for every module $m \in M$. The value d_m represents the rank of the module in the line (from 1 to n), moving 'backward' along the $\overrightarrow{Y}$ axis starting from module m, until an empty position is reached.

To compute d_m, the algorithm proceeds as follows (see example Fig. 2a) :

1) Initially, modules without an attached neighbor in the backward direction must set d to 1.
2) Then, they send a message, denoted as `SET_D_MSG(`d`)`, to their front neighbor. Upon receiving the message, the module sets its d value

Algorithm 1: Distributed shape recognition - Part 1

input: (x, y, z), neighbors

Initialization:
  if $empty((x, y+1, z))$ **then**
    send SET_D_MSG(1) to $neighbor((x, y-1, z))$
Msg Handler *SET_D_MSG(d_{sent})*:
  $d \leftarrow d_{sent} + 1$
  if $\neg empty((x, y-1, z))$ **then**
    send SET_D_MSG(d) to $neighbor((x, y-1, z))$
  else
    if *isRmin* **then**
      if $\neg empty((x+1, y, z))$ **then**
        send FIND_W_MSG(id, d) to $neighbor((x+1, y, z))$
      else
        $w = 1$
        Notify front line of w
        if $\neg empty((x, y, z+1)$ **then**
          send FIND_H_MSG(id, d, w) to $neighbor((x, y, z+1))$
        else
          myBox = $(\{x, y, z\}, \{x+w-1, y+d-1, z+h-1\})$

Algorithm 2: Distributed shape recognition - Part 2

```
1  Msg Handler FIND_W_MSG(id, d_sent):
2    if d < d_sent then
3      send SET_W_MSG(id, d_sent, 0) to
         neighbor((x, y − 1, z))
4    else
5      if empty((x + 1, y, z)) then
6        w ← 1
7        Notify back line of w
8        send SET_W_MSG(id, d_sent, w) to
           neighbor((x, y − 1, z))
9      else
10       send FIND_W_MSG(id, d_sent) to
           neighbor((x + 1, y, z))
11 Msg Handler SET_W_MSG(id, d_sent, w_sent):
12   if d_sent ≥ d then
13     w ← w_sent + 1
14     Notify back line of w
15     if myid = id then
16       if ¬empty((x, y, z + 1)) and isCmin) then
17         send FIND_H_MSG(id, d, w) to
             neighbor((x, y, z + 1))
18       else
19         myBox = ({x, y, z}, {x + w − 1, y + d − 1, z})
20   else
21     send SET_W_MSG(id, d_sent, w_sent + 1) to
         neighbor((x, y − 1, z))
22 Msg Handler FIND_H_MSG(id, d_sent, w_sent):
23   if w < w_sent ∨ d < d_sent then
24     send SET_H_MSG(id, 0) to neighbor((x, y, z − 1))
25   else
26     if empty((x, y, z + 1)) then
27       h ← 1
28       send SET_H_MSG(id, h) to
           neighbor((x, y, z − 1))
29     else
30       send FIND_H_MSG(id, d, w) to
           neighbor((x, y, z + 1))
31 Msg Handler SET_H_MSG(id, h_sent):
32   h ← h_sent + 1
33   if myid = id then
34     myBox =
         ({x, y, z}, {x + w − 1, y + d − 1, z + h − 1})
35   else
36     send SET_H_MSG(id, h) to
         neighbor((x, y, z − 1))
```

as the received value plus one and, subsequently, forwards the message to its front neighbor.

3) This process continues until an empty position is encountered in the front direction (cf. Algorithm 1 lines 1-7).

Once d_m is set, module m can determine locally if it is R_{min}. We denote as d_{left} the d value of the module on the left of the module m. Then, a module is R_{min} if it verifies the following condition:

$$empty(front) \wedge (d \neq d_{left} \vee \neg empty(left + front)) \quad (2)$$

Next, we define w_m for each module m in the set M as the maximum number of connected vertical lines with the same or higher height toward the right (cf. Fig.2b).

We can express w_m by the following rule:

$$w_m = \max_{n \text{ connected}} (y_m = y_n \wedge (d_i \geq d_m \forall i \in [m, n])) \quad (3)$$

Once the value of w_m is determined, we can refer to the values of d and w of the module at the bottom of module m as d_{bottom} and w_{bottom}, respectively. We also use $R_{min}(bottom)$ to indicate whether the bottom module is at an R_{min} position. Then a module m is at a C_{min} position if it verifies:

$$R_{min} \wedge \neg (R_{min}(bottom) \wedge d = d_{bottom} \wedge w = w_{bottom}) \quad (4)$$

To determine w_m for all modules m in the set M, the following distributed process is employed: each module located in a position R_{min} sends a message called $\texttt{FIND\_W\_MSG}(id, d_{sent})$ to its right neighbor (cf. Algorithm 1 lines 9-10). As the message is forwarded, if the message reaches a module n with $d_n < d_{sent}$, the module n responds to the sender on its left with a message $\texttt{SET\_W\_MSG}(id, d_{sent}, w_{sent} = 0)$. Otherwise, when it reaches a module n that lacks a right neighbor, it assigns w_n the value of 1 and responds to the sender on the left using

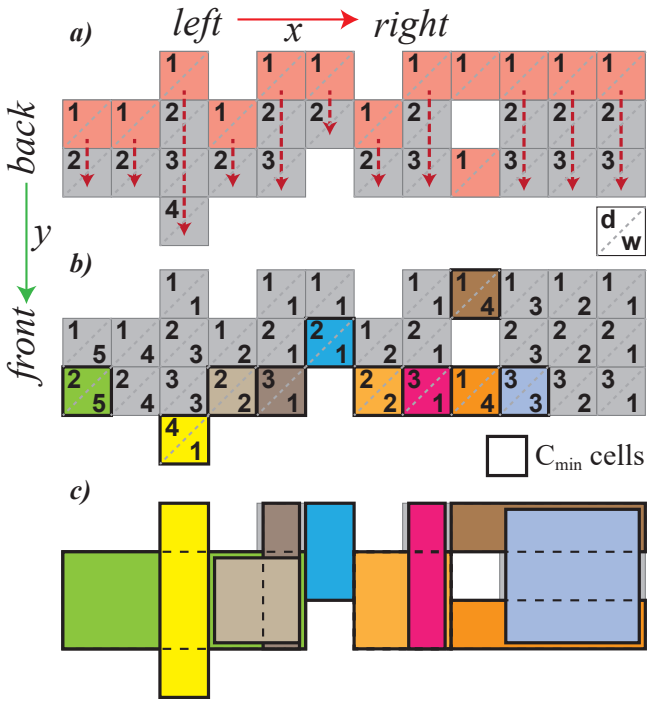


Fig. 2: Distributed computation of the boxes on 2D. a) Computation of vertical distance d. b) Election of C_{min} cells and computation of w. c) Boxes covering the plane using C_{min} cells colors.

a message denoted as SET_W_MSG$((id, d_{sent}, w_{sent} = 1)$. Upon receiving this message, module r sets its w_r value as the received value plus one if and only if $d_{sent} \geq d_r$ to ensure that the line at the back of module r can accommodate d_{sent} modules (cf. Algorithm 2 lines 1-19).

From each R_{min} module, we use local d_m and w_m values to define a rectangle $(R_{min}, R_{max}) = (\{x_m, y_m\}, \{x_m + w_m - 1, y_m + d_m - 1\})$. This approach enables the creation of a vertical decomposition consisting of overlapping rectangles on each 2D layer along the $\overrightarrow{Z}$ axis.

The next step consists in determining the height h, which represents the maximum number of rectangles on top of the one associated to C_{min}. Consequently, this results in the formation of overlapping boxes. To accomplish this, the module located at C_{min} initiates a message called FIND_H_MSG$(id, d_{sent} = d, w_{sent} = w)$. This message serves to count the number of top neighbors along the $\overrightarrow{Z}$ axis that meet conditions $d \geq d_{sent}$ and $w \geq w_{sent}$ (cf. Algorithm 2 lines 16-19).

Upon receiving the FIND_H_MSG message by a module r, if the values $d_r > d_{sent}$ or $w_r > w_{sent}$, it replies with a message SET_H_MSG(id, h=0) to the sender at the bottom. Otherwise, if a module r lacks a top neighbor, it responds by sending a message labeled SET_H_MSG(id, h=1) to the sender at the bottom. Upon receiving the SET_H_MSG message, the receiver increments the received h value by one and forwards the message to its bottom neighbor until reaching the initiator (cf. Algorithm 2 lines 22-36).

The algorithm operates asynchronously. Therefore, a module can receive a FIND_W_MSG before its d value is defined or a FIND_H_MSG before the d and w values are defined. To solve this, if a module receives a message and the values required for its handling are not yet defined, the module stores the received message in its memory and handles it once the values are set.

When module m at a C_{min} receives its h_m value it can set its Box as $(\{x_m, y_m, z_m\}, \{x_m + w_m - 1, y_m + d_m - 1, z_m + h_m - 1\})$. The algorithm terminates when all modules at a C_{min} position have determined their boxes.

Fig. 3 shows an example of overlapping boxes. It can be seen that the resultant boxes may differ according to the orientation of the coordinate axis. However, in both cases a) and b) the resultant boxes cover all the configuration. Moreover, having an overlapping box results in boxes that are completely inside a bigger one, such as the blue and purple in Fig. 3 (a) and the yellow in Fig. 3 (b). When multiple boxes must be stored, they can be aggregated by neglecting the boxes that are inside another one.

IV. Complexity Analysis

The complexity in terms of the number of boxes has a lower bound of $\Omega(1)$ in the case of a cubic configuration. As the shape becomes increasingly irregular and incorporates holes, the complexity approaches an upper bound of $O(n)$ where n is the number of modules.

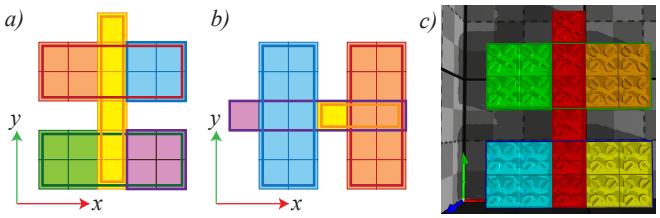


Fig. 3: Three basic example: a) A basic shape defined by 5 overlapping boxes. b) The same shape but rotated producing 4 boxes only. c) The same shape computed on *VisibleSim*.

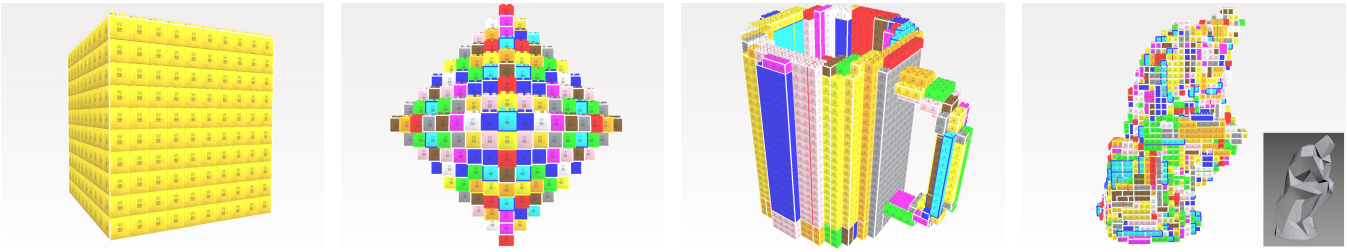
Next, we assess the time and communication complexities of the shape recognition algorithms presented in Section III. The number of messages used to find the boxes is proportional to the number of modules. To find the the values of d and w of each module on a line along the $\overrightarrow{X}$ and $\overrightarrow{Y}$ axes, $O(n)$ messages are exchanged, where n is the number of modules. Then, to find the height of each box, the box's corner C_{min} initiates a message that passes to the line of modules along the $\overrightarrow{Z}$. The number of these messages is also bounded by $O(n)$. Therefore, the communication complexity can be expressed as $O(n)$.

As for the time complexity, the search for boxes is done in parallel. Setting the values of d and w requires $O(D + W)$ time, where D and W are the depth and width of the entire configuration. The time complexity to search for the height h of the boxes is $O(H)$, where H is the height of the configuration. Consequently, the overall time complexity can be expressed as $O(D + W + H)$. Thus, it depends on the geometry of the configuration.

V. Experiments and Analysis

We implemented the algorithm using *VisibleSim*, a discrete event-based simulator for distributed modular robotic systems that support *Blinky Blocks* [13]. Fig. 3c shows a *VisibleSim* capture of the simulated example of the box cover shown in Fig. 3. *Blinky Blocks* system is a modular robotic system made up of centimetre-size blocks that are attached to each other via magnets in a square cubic lattice. Each block is a cube of roughly $40\,\mathrm{mm}$, with processing, storage, and communication capabilities. Each *Blinky Block* communicates through serial links with its directly connected neighbors by sending packets with a payload size of 227 bytes.

The objective of the experiment is to compare the shape recognition algorithm with a coordinates collection method in order for modules to send their current shape to a computer to be used by an interactive CAD software. The coordinates collection method consists of sending the list of coordinates to the root of a breadth-first spanning tree connected to the CAD computer. Each module sends three bytes for its coordinates x, y and z. The leaf modules start by sending their coordinates. The coordinates are merged at intermediate modules before being sent to their parent in the tree when the data are received from all their children modules or the payload is totally used. Using the shape recognition method,



(a) (b) (c) (d)

Fig. 4: Four configurations examples captured from *VisibleSim*. (a) Cube configuration with 1000 *Blinky Blocks*. (b) Ball configuration with radius 8 (833 *Blinky Blocks*). (c) Mug configuration with 4019 *Blinky Blocks*. (d) Thinker configuration with 5814 *Blinky Blocks*.

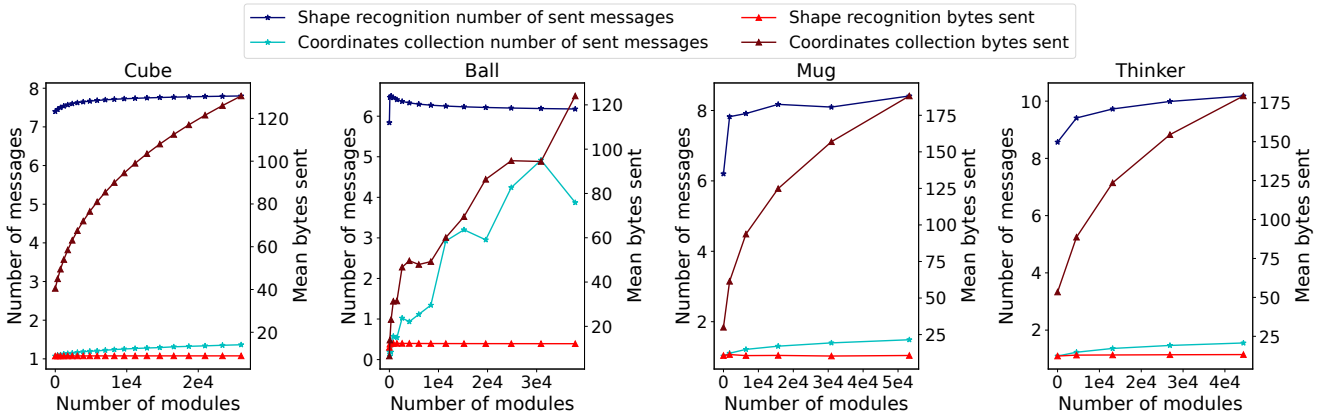


Fig. 5: Mean number of messages and mean number of bytes sent per module on the four configurations examples.

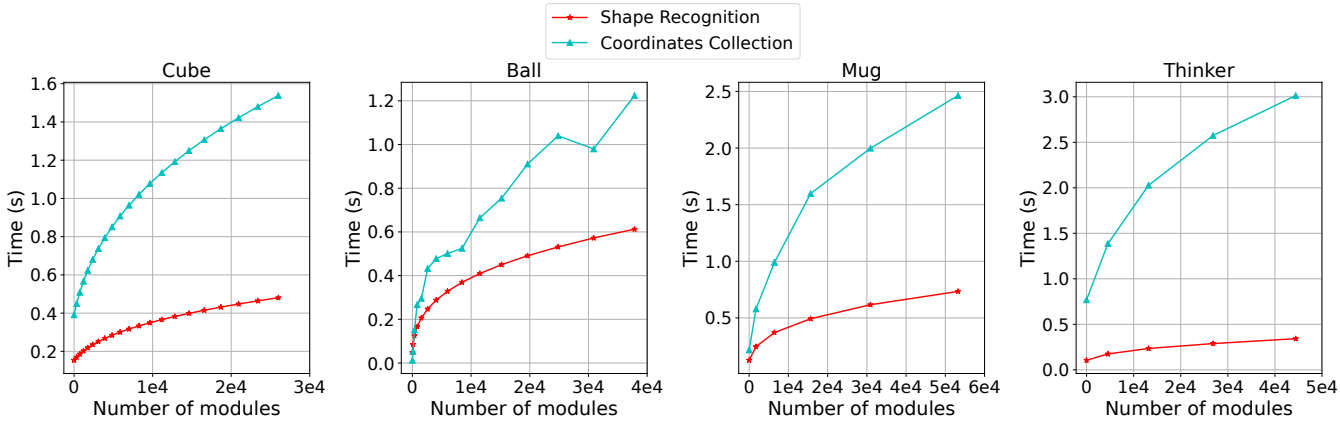


Fig. 6: Time for receiving the whole current shape by the root.

once a box is found, the module at C_{min} sends the box information to the root also via a breadth-first tree.

We have done the comparison on four different configurations shown in Fig. 4 with different geometries and characteristics:

1) Cube: A simple and regular connected cubic shape. One box is required to cover the whole configuration.
2) Ball: It contains modules whose distance from the center is less than or equal to a given radius [14].
3) Mug: A mug shape that exhibits a few irregularities.
4) Thinker: The thinker statue defined by a low resolution mesh.

We evaluated the mean number of messages sent per module, the mean number of bytes sent by a module, the time taken to complete the shape recognition, and the ratio between the number of modules and the number of boxes while increasing the sizes of the configurations.

Fig. 5 shows the mean number of messages and the mean number of bytes sent by a module on the four configurations. The mean number of messages sent by a module executing

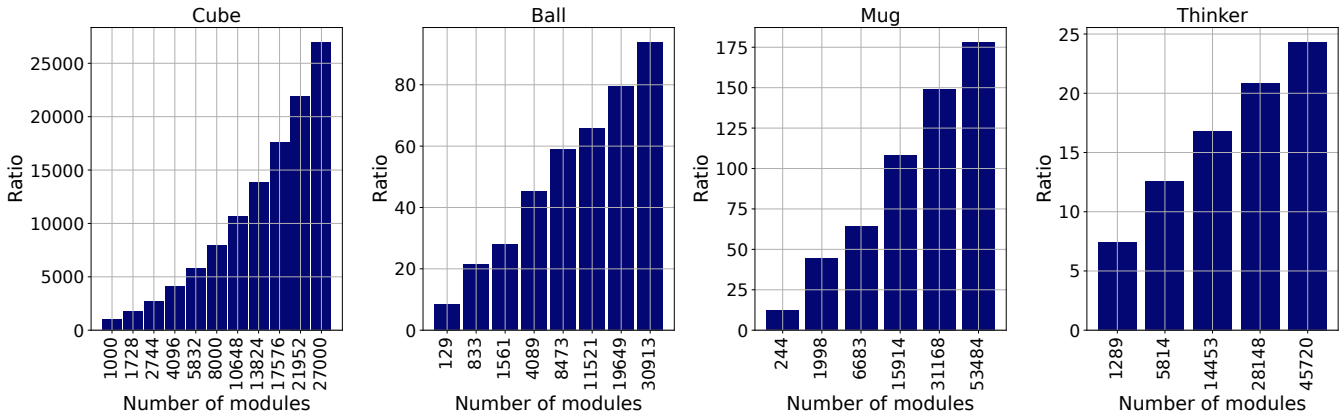


Fig. 7: Ratio between the number of modules and the number of boxes.

the shape recognition method is larger than the coordinates collection on the four configurations. This is due to the communication required to find the dimensions of the boxes. As for the coordinates collection method, each module must send one message to its parent that contains the coordinates of its subtree, but due to the limitation of the packet size, when a packet is filled, it is directly sent to the root, which increases the number of messages sent by a module. More packets will be full as the size of the configuration increases. Regarding the mean number of bytes sent by a module, the sizes of the messages exchanged by the shape recognition algorithm to find the boxes are limited. The maximum size of a used message is 6 bytes which contain the coordinates of the C_{min} and C_{max} of the box. For the coordinates collection method, the mean message size increases with the size of the configuration. The maximum packet size can go up to its payload capacity.

We conducted an experimental study on *Blinky Block* hardware that showed that the time t required per message is affected by the message length l (number of bytes contained in a message) and can be modeled with the linear function: $t = 0.08935 \times l + 1.516$. Therefore, the global executed time is affected by the number of exchanged messages and the length of the messages. Fig. 6 shows the time taken by both methods. Although the shape recognition method requires more messages, it can be seen that the shape recognition method is more efficient in time in the four configurations due to the increase in the length of the messages used in the coordinated collection method as the configuration size increases. The time taken by the shape recognition method is independent of the number modules. It depends on the geometry of the configuration as explained in Section IV.

Fig. 7 illustrates the ratio between the number of modules and the number of boxes in different configurations. The cube shape represents the most favorable scenario, as all modules can be accommodated within a single box. As the configuration becomes more irregular, such as in the case of the thinker configuration, the ratio decreases. This decrease is a consequence of the need for additional boxes with smaller dimensions to accommodate the irregularities present in the structure.

VI. Conclusion and Future Works

In this paper, we proposed a new shape recognition algorithm that allows modules in a lattice-based modular robot to discover their current shape. The modules search for overlapping boxes to cover the whole configuration. The union of these boxes gives the current shape. We evaluated the algorithm in simulation on *Blinky Blocks* on different configurations with different geometrical properties and compared it with a coordinates collection method to retrieve the current shape of the ensemble and send it to a central entity. The results show that the shape recognition method outperforms the coordinates collection in time efficiency while using a smaller memory footprint.

In the future, our work will focus on developing a dynamic version of the algorithm that allows the modules to keep track of their shape in real time. We also aim to work on a distributed aggregation method for boxes to minimize their number by ignoring boxes that are fully included in larger ones. Plus, we will investigate the possibility to use other geometrical shapes, in addition to the simple cubic box shape which reduces the number of elements that cover the configuration.

ACKNOWLEDGMENT

This work has been supported by the EIPHI Graduate School (contract "ANR-17-EURE-0002").

References

[1] J. Bourgeois, B. Piranda, A. Naz, N. Boillot, H. Mabed, D. Dhoutaut, T. Tucci, and H. Lakhlef, "Programmable matter as a cyber-physical conjugation," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, oct 2016, pp. 002 942–002 947. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02140347http://ieeexplore.ieee.org/document/7844687/

[2] 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

[3] B. Piranda, P. Chodkiewicz, P. Hołobut, S. P. A. Bordas, J. Bourgeois, and J. Lengiewicz, "Distributed prediction of unsafe reconfiguration scenarios of modular robotic programmable matter," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2226–2233, 2021.

[4] T. Tucci, B. Piranda, and J. Bourgeois, "Efficient scene encoding for programmable matter self-reconfiguration algorithms," *Proceedings of the ACM Symposium on Applied Computing*, vol. Part F1280, pp. 256–261, 2017.

[5] J. Baca, B. Woosley, P. Dasgupta, and C. Nelson, "Real-time distributed configuration discovery of modular self-reconfigurable robots," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, no. June. IEEE, may 2015, pp. 1919–1924. [Online]. Available: http://ieeexplore.ieee.org/document/7139449/

[6] M.-C. Shiu, L.-C. Fu, and Y.-J. Chia, "Graph isomorphism testing method in a self-recognition velcro strap modular robot," in *2010 5th IEEE Conference on Industrial Electronics and Applications*. IEEE, 2010, pp. 222–227.

[7] C. Liu and M. Yim, "Configuration Recognition with Distributed Information for Modular Robots," in *Springer Proceedings in Advanced Robotics*. springer, 2020, vol. 10, pp. 967–983. [Online]. Available: http://link.springer.com/10.1007/978-3-030-28619-4-65

[8] Z. Butler, R. Fitch, D. Rus, and Yuhang Wang, "Distributed goal recognition algorithms for modular robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 1. IEEE, 2002, pp. 110–116. [Online]. Available: http://ieeexplore.ieee.org/document/1013347/

[9] K. Stoy and R. Nagpal, "Self-Reconfiguration Using Directed Growth," in *Distributed Autonomous Robotic Systems 6*. Tokyo: Springer Japan, 2008, pp. 3–12. [Online]. Available: http://link.springer.com/10.1007/978-4-431-35873-2-1

[10] R. Fitch and Z. Butler, "Million module march: Scalable locomotion for large self-reconfiguring robots," *International Journal of Robotics Research*, vol. 27, no. 3-4, pp. 331–343, 2008.

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

[12] 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.

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

[14] A. Naz, B. Piranda, T. Tucci, S. Copen Goldstein, and J. Bourgeois, "Network characterization of lattice-based modular robots with neighbor-to-neighbor communications," in *Distributed Autonomous Robotic Systems: The 13th International Symposium*. Springer, 2018, pp. 415–429.