A Fast Distributed Algorithm for Breakage Detection in Modular Robots

Ikrame Yazidi[1], Lucas Berthome[2], Morvan Ouisse[2][0000−0001−5464−5283], and Benoit Piranda[1][0000−0003−2149−871X]

[1] Marie and Louis Pasteur University, FEMTO-ST institute, CNRS Monbéliard, France.
benoit.piranda@umlp.fr

[2] Marie and Louis Pasteur University, SUPMICROTECH, FEMTO-ST institute, CNRS, Besançon, France.
morvan.ouisse@umlp.fr

Abstract. In this paper, we present a fully distributed method that allows a group of magnetically connected modular robots to assess the mechanical stability of their structure. Stability is verified based on four mechanical phenomena: vertical and rotational sliding and rotational debonding for vertical and lateral connectors. We present a mechanical model applicable to the *Blinky Blocks* robot system. We also propose an efficient, fully distributed algorithm that runs simultaneously on all robots, enabling them to check global stability and detect the type and the breakage positions. Our algorithmic solution avoids the global resolution of the system and is effective for both free-loop and complex loop or multi-loop configurations; for the latter, it systematically enumerates all possible spanning trees and traverses each one to assess the stability of the modular structure. The approach was validated both in simulation with *VisibleSim* and on real *Blinky Blocks* hardware. Experiments demonstrate accurate and robust detection of all four breakage types in a wide variety of configurations, with low computational overhead and excellent scalability. These results confirm that distributed spanning-tree-based analysis provides an effective alternative to solving global equilibrium equations in modular robotics, enabling reliable structural integrity assessment even in large and highly connected assemblies.

1 Introduction

Over the past 20 years, many studies have introduced the concept of programmable matter [3,1] as a solution for creating materials with new properties, such as changing color or shape on command from an integrated program. They have proposed hardware techniques and software solutions for creating the first self-reconfigurable robot systems [9,8,10,5,6].

Programmable matter consists of small, reconfigurable modular robots attached to one another in a regular grid. The concept of modular robots is important because it introduces the concept of autonomy: it is the elements that

make up the material itself that modify an attribute such as their color or reorganize themselves to change the overall shape they construct. The first challenge proposed by Programmable matter is its requirement for small robots as they define the resolution of the objects they construct.

This brings us to the second major challenge raised by the concept of programmable matter, which is the issue of distributed programming. In order to establish consistent emergent behaviors among all robots, they must communicate and synchronize their actions. This large group of robots will therefore build a network and an associated graph that executes its processes within its nodes: the robots. The solution is provided by distributed programming algorithms, often carried out by distributed agents.

Despite their promising potential, a challenge remains: creating robust systems and algorithms that can respond to unforeseen situations caused by communication delays, missed travel deadlines, or module failures. Mechanical stability is a major concern for system robustness. Even if the robots are perfectly functional and the reconfiguration algorithm perfectly organizes their movements, the robots are subject to the mechanical stability of the system they build. Consequences of instability can include breaking an attachment link between modules, cutting the network between two modules due to excessive deformation of the link, or the system falling and deteriorating, which jeopardizes any future movement.

The majority of algorithms for planning and controlling the self-reconfiguration of modular robots assume that all motions verify stability criteria. Early work [7] proposed precise but iterative solutions that required excessive computation time and message counts for a system to use them in an auto-reconfiguration process. In [2], Bray and Groß used a distributed control strategy to reconfigure self-assembled structures that fill a void, while incorporating local force measurements to build structures that do not disintegrate under their own weight. However, autonomous reconfiguration planning that considers mechanical constraints remains an open and complex problem. A more recent work [13,12] has proposed dealing only with the validation of the balance of a configuration placed on the ground. This decision is made quickly by running through all the modules once. Three static mechanical phenomena will be addressed: slippage, tearing, and detachment. Our goal is to incorporate this tool into our auto-reconfiguration algorithms to verify structural stability before moving a module. Therefore, this process must be fast enough to evaluate multiple potential movements.

To illustrate the effectiveness of our approach, we will present several experiments using different numbers of modules. The verification is carried out in a dedicated simulator *VisibleSim* [11], as well as experimentally on real robotic modules *Blinky Blocks* [4].

2 Context

Blinky Blocks are centimeter-sized modular robots that are attached via permanent magnets (cf. Figure 1(a)). Each block, approximately a 41 mm large cube,

Table 1. Quantities used in the paper.

Quantity	Parameter	Value	Variable	Expression	Value
Blinky Block width	a	41 mm	T_a	$\frac{f F_{magL}}{mg}$	14.5
Mass	m	61.2 g	T_b	$\frac{A f F_{\mathrm{magL}}}{6mg}$	4.77
Gravity	g	9.81 ms^{-2}	T_c	$\frac{h F_{\mathrm{magL}}}{mga}$	13.4
Friction coefficient	f	0.5	T_d	$\frac{F_{\mathrm{magV}}}{2mg} - \frac{1}{2}$	9.47
Rotation center height	h	2.15 cm			
Magnetic force (side)	F_{magL}	15.0 N			
Magnetic force (top)	F_{magV}	12.0 N			

has computing power, sensors, actuators and inter-block communication capabilities. All blocks execute the same program and communicate via a neighbor-to-neighbor model. The embedded program can detect neighborhood evolutions in a robot, enabling real-time adaptations based on new physical arrangements. For the purpose of detecting breakage, we consider that the *Blinky Block* placed on the ground is fixed to it, and that they are all subject only to the vertical force of gravity, magnetic and contact forces between *Blinky Blocks*. The magnetic forces produced by the magnets on the lateral and vertical faces, as well as the friction forces, were measured experimentally. All the data used in this article are specified in Table 1.

Our mechanical study is based on the use of the fundamental law of static. A mechanical system is in static equilibrium if and only if the sum of the forces and the sum of the moments acting on it are equal to zero. This can be expressed mathematically as:

$$\sum \overrightarrow{F_i} = 0 \tag{1}$$

where $\overrightarrow{F_i}$ represents the forces applied to the body. Additionally, for rotational equilibrium, the sum of the moments must also be zero:

$$\sum \overrightarrow{M_i} = \sum \overrightarrow{r_i} \times \overrightarrow{F_i} = 0 \tag{2}$$

Here, $\overrightarrow{M_i}$ represents the moments to which the body is subjected, about a point. The forces at play are gravity, contact and magnetic forces that hold the cubes together. We present here the four types of breakage phenomena addressed in this paper:

a) The vertical sliding is produced by the vertical motion of the group H in green in Figure 1(b).a along the vertical plane P_a and following the vertical axis $-\overrightarrow{z}$.
b) The rotational sliding is the rotation of the group H around the interface face between G and H.
c,d) The rotational debounding can appear in a vertical interface plane (Figure 1(b).c) or in an horizontal interface plane (Figure 1(b).d), the H group turns around the red axis.

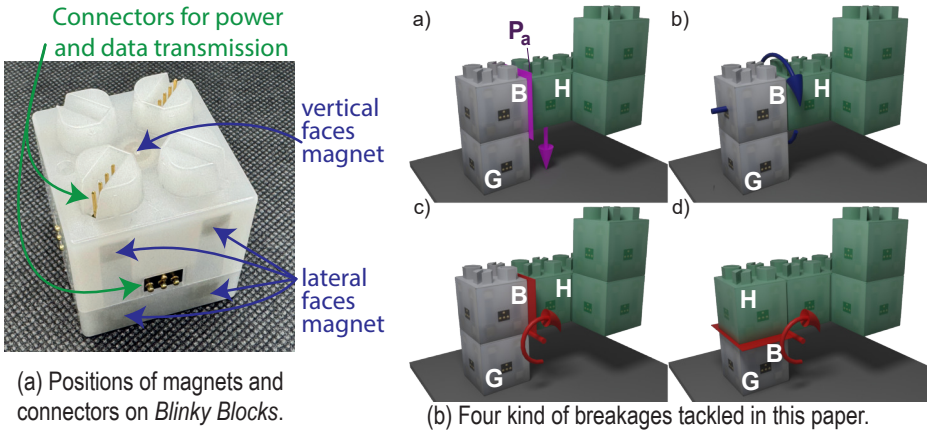


Fig. 1. *Blinky Block* Hardware and possible brackages.

For all these phenomena, we consider a fixed G part attached to the ground represented by gray *Blinky Blocks* in Figure 1(b) and the group of *Blinky Blocks* H that can be separated to G due to external forces. We call B the *Blinky Block* of G whose one face forms the interface between G and H.

To establish the vertical sliding criterion for a lateral connection, we consider the system in static equilibrium at the sliding limit. According to Equation 1 we can write the static resultant, along to z axis (cf. Figure 2(b)):

$$-n_H\, mg + T = 0 \tag{3}$$

Where n_H is the number of blocks of H and T is the tangential friction force. At the sliding limit, $T = fN$ with f the coefficient of friction and N the force normal to the faces at the link. In the case of a loop-free structure, $N = F_{\text{magL}}$ is the magnetic force on one side of a cube. Therefore, for sliding to occur, the weight of the cantilever cubes must be greater than the tangent friction force, which means that:

$$n_H > \frac{f F_{\text{magL}}}{mg} \tag{4}$$

Considering our *Blinky Blocks*, $T_a = \frac{f\, F_{magL}}{mg} \approx 14.5$, then we have to check if N_H, the number of connected modules in H, is greater than 14.

About rotational sliding criterion (lateral detachment), the focus is not on vertical sliding, but on rotational sliding around the axis normal to the link faces and passing through their center A, as illustrated in Figure 2(b). The total moment of friction $\overrightarrow{M_A}$, is the sum of the components of all the points M, which are defined by the $\overrightarrow{AM}$ polar coordinates (r, θ) of the face:

$$\overrightarrow{M_A} = \iint_{\text{face}} \overrightarrow{AM} \wedge \mathrm{d}\overrightarrow{T}(r, \theta) \tag{5}$$

Using a symbolic method to simplify this equation, we get: $M_A = \frac{BfaF_{\text{magL}}}{6}$. The static moment theorem yields:

$$mga\left|\sum_{i\in H}(y_i - y_A)\right| = \overrightarrow{M_A}.\overrightarrow{e_x} \Rightarrow \left|\sum_{i\in H}(y_i - y_A)\right| = \frac{BfF_{\text{magL}}}{6mg} \tag{6}$$

With y_i the non-dimensional y coordinate of the center of gravity of *Blinky Block* i and y_A the non-dimensional y coordinate of the center of reference face A (i.e. (x_i, y_i) are the cell coordinates in the grid). The physical coordinates write $Y_i = ay_i$. Setting $T_b = \frac{BfF_{\text{magL}}}{6mg}$, we deduce that rotational sliding occurs if:

$$\left|\sum_{i\in H}(y_i - y_A)\right| > T_b \text{ or } \left|\sum_{i\in H}(x_i - x_A)\right| > T_b \tag{7}$$

Considering the rotational debonding criterion on a lateral face $(0, \overrightarrow{z}, \overrightarrow{x})$ with normal vector $+\overrightarrow{x}$, Equation 8 holds when the system is such that the contact force cancels.

$$-mga\left|\sum_{i\in H}(x_i - x_A)\right| + hF_{\text{magL}} = 0 \tag{8}$$

With x_A the non-dimensional positions of the center of face A. Setting $T_c = \frac{hF_{\text{magL}}}{mga}$, with $h = \frac{a}{2}$ we can deduce that a rotational debonding occurs if:

$$\left|\sum_{i\in H}(x_i - x_A)\right| < T_c \text{ or } \left|\sum_{i\in H}(y_i - y_A)\right| < T_c. \tag{9}$$

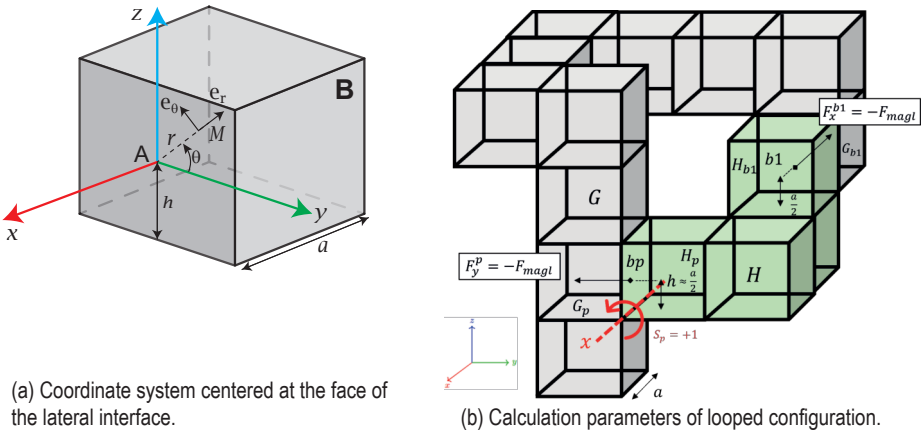
Similar conditions can be derived for rotational debonding criterion relatively to the vertical interface for a face with normal vector $+\overrightarrow{z}$: We define $T_d = \frac{F_{magV}}{2mg} - \frac{1}{2}$. Considering a connection to a top or a bottom interface, the integrity is not ensured if :

$$\left|\sum_{i\in H} x_i - x_A\right| < T_d \quad \text{or} \quad \left|\sum_{i\in H} y_i - y_A\right| < T_d. \tag{10}$$

Creating only loop-free structures is restrictive, particularly in terms of mechanical integrity. For the generalization to 3D looped configuration, we consider p a point placed anywhere along the rotation axis. For example, in the Figure 2(b), p can be placed anywhere on the red axis x since the breakage condition around this axis does not depend on x coordinates. This figure also shows the parameters used in the following stability-condition equations, F^{bj} is the magnetic force apply to the face b of block j.

The breakage condition around the x-axis is expressed as:

$$s_p\left[M_{bpx} + \sum_{j=1}^{n}\left[(y_{bj} - y_p)F_z^{bj} - (z_{bj} - z_p)F_y^{bj}\right]\right] < 0 \tag{11}$$



(a) Coordinate system centered at the face of the lateral interface.

(b) Calculation parameters of looped configuration.

Fig. 2. Geometrical parameters for the mechanical expressions.

Similary, for a y-axis:

$$s_p \left[M_{bpy} + \sum_{j=1}^{n} \left[(z_{bj} - z_p) F_x^{bj} - (x_{bj} - x_p) F_z^{bj} \right] \right] < 0 \tag{12}$$

With M_{bp} represents the moment of gravity.

When the pivot joint is mounted on a lateral face, sliding conditions and potential rotation must be checked by:

$$|N_H mg| > f \left(|F_{magL}| + \sum_{j=1}^{n} \left| F_x^{bj} \right| + \sum_{j=1}^{n} \left| F_y^{bj} \right| \right) \tag{13}$$

The conditions for a rotational debonding around the axis normal to the face:

– For a normal x face:

$$\left| -mga \sum_{i \in H} (y_i - y_{bp}) + \sum_{j=1}^{n} \left[(y_{bj} - y_{bp}) F_z^{bj} - (z_{bj} - z_{bp}) F_y^{bj} \right] \right| > |M_A| \tag{14}$$

– For a normal y face:

$$\left| -mga \sum_{i \in H} (x_i - x_{bp}) + \sum_{j=1}^{n} \left[(z_{bj} - z_{bp}) F_x^{bj} - (x_{bj} - x_{bp}) F_z^{bj} \right] \right| < |M_A| \tag{15}$$

To sum up, for each G-H separation, we must check whether the pivot joint is vertical or lateral. If it is **vertical**, we need to verify equations 11 and 12. If it is **lateral**, we must also verify equations 13, 14 and 15. Therefore, for each

cut, the pivot orientation must be changed, and the necessary checks must be repeated. **Vertically**, there are **four possible rotation axes**: two along the x-axis and two along the y-axis. **Laterally**, there are only **two possible pivot** along the z-axis.

3 Algorithmic Contribution

The proposed algorithms are a fully distributed, tree-based partitioning algorithms designed to efficiently handle a substantial number of modules. they provide scalability by handling communication between neighbouring modules efficiently, meaning that modular robots can interact and collaborate autonomously to solve the stability problem.

If we study the models associated with the four breakage phenomena (Equations 4, 7, 9, and 10), we notice that they can be written as Boolean functions that only depend on $\sum x$, $\sum y$ and N_H. This property allows us to define the break detection algorithm in two steps: we construct a spanning tree from the *Blinky Block* fixed to the ground by performing a BFS from this block and then traversing the tree from the leaves, we determine the values $\sum x$, $\sum y$ and N_H for each module. Once these quantities are defined, the above conditions are applied to validate the stability of the subtree with respect to each internal node. Once the root is reached, the stability of the entire configuration can be concluded.

Considering a reference frame centered on the interface module B, we define S_x^c as the sum of the distances along the axis $\overrightarrow{x}$ between the centers of the modules of H and the center of module B:

$$S_x^c = \sum_{i \in H} x_i - x_B = \sum_{i \in H} x_i - N_H \times x_B \tag{16}$$

But if we now consider a reference point centered on the normal face $\overrightarrow{x}$ at the interface between B and H (as in Figure 2(b)), we define the distance S_X^f:

$$S_x^f = \sum_{i \in H} x_i - x_O = \sum_{i \in H} x_i - N_H \times (x_B + 0.5) \tag{17}$$

and if the normal is $-\overrightarrow{x}$, we get:

$$S_{-x}^f = \sum_{i \in H} x_i - x_O = \sum_{i \in H} x_i - N_H \times (x_B - 0.5) \tag{18}$$

Using these tools, we define the four boolean functions that are true if a breakage occurs. We also use the quantities shown in Table 1 to verify these functions:

$$\begin{aligned}
F_a &= (N_H > T_a) \\
F_b &= \left((face_{+\overrightarrow{x}} \vee face_{-\overrightarrow{x}}) \wedge \left(\left|S_y^c\right| > T_b\right) \vee \left((face_{+\overrightarrow{y}} \vee face_{-\overrightarrow{y}}) \wedge \left(|S_x^c| > T_b\right)\right)\right. \\
F_c &= \left((face_{+\overrightarrow{x}} \vee face_{-\overrightarrow{x}}) \wedge \left(\left|S_y^f\right| < T_c\right) \vee \left((face_{+\overrightarrow{y}} \vee face_{-\overrightarrow{y}}) \wedge \left(\left|S_x^f\right| < T_c\right)\right)\right. \\
F_d &= \left(\left(\left|S_x^f\right| < T_d\right) \vee \left(\left|S_y^f\right| < T_d\right)\right)
\end{aligned} \tag{19}$$

Algorithm 1: Enumeration of Spanning Trees

```
1  Function enumerateSpanningTrees(T_0 , A_p):
2      Push initial tree T_0 onto stack S
3      Add T_0 to list of visited trees
4      while S! = ∅ do
5          T ← pop(S)
6          foreach edge a ∈ A_p do
7              if a ∉ T then
8                  T_a ← T ∪ {a}
9                  cycle ← findCycle(T_a, a)
10                 foreach edge f ∈ cycle | f ≠ a do
11                     T_new ← T ∪ {a} \ {f}
12                     if reverse(f) ∈ T_new then
13                         Remove reverse(f) from T_new
14                     if T_new ∉ visited then
15                         Add T_new to visited
16                         push T_new into S
```

Our algorithm is based on the propagation of messages between modules via a dynamically built spanning tree from a leader. At startup, the leader triggers the propagation of a “Request” message to all its neighbors, initiating the construction of a spanning tree. When a module receives a "Request" message for the first time, it designates the sender as its parent. Then, it sends “Requests” to all its own neighbors. If this module is a leaf in the tree, so it sends a "Result" message directly to its parent. A handler is called when a module receives a "Result" response from one of its child modules.

The module checks for rotational, lateral, or vertical breakage, as well as slippage, using the data sent by its child. The module sums up the positions and number of nodes of its children. Once all the expected responses have been received, the module adds its own position to the sum and returns the "Result" message to its parent, continuing the propagation of results to the tree root. This continues the propagation of results to the tree root.

The proposed algorithm for looped configurations is an algorithm in which each module collaborates with its neighbors to build and explore all possible spanning trees of a graph representing the network of connected modules. In looped configurations, there are several possible spanning trees. The aim is to build and explore them one by one. The first step is to build the basic spanning tree the same way as the free-loop algorithm. The leader initiates the propagation of a broadcast message. When a module receives this message for the very first time, it considers the sender to be its parent. At this point, it adds the edge linking it to its parent to a list called $T0$. This $T0$ list will contain all the edges of the basic spanning tree. It is built up progressively as modules connect to each other through the propagation of the initial message. When a module receives other broadcast messages after the first one, the module doesn't change parent, but takes note that other connections are possible. The edge between the

Algorithm 2: Traversal from leaf to root and stability checking

```
1  Function traverseToRoot(start, adj, position, parents):
2      visited ← ∅
3      sumH ← (0, 0, 0)
4      push start in q1
5      nh ← 0
6      while q1 ! = ∅ do
7          node ← pop(q1)
8          if node ∉ visited then
9              visited ← visited ∪ node
10             nh ← nh + 1
11             sumH ← sumH + position[node]
12             if node has a parent in parents then
13                 parent ← parents[node]
14                 Fp ← computeMagneticForce(position[node], position[parent])
15                 check breakage for different cases
16                 if parent ∉ visiteds then
17                     push parent into q1
```

module and these other senders is added to another list called *Ap*. This *Ap* list contains all the edges that are part of the basic tree, as well as those that represent valid connections between modules. These are known as additional edges and are useful for generating other spanning trees. If the module configuration contains no cycles, each module has only one path to reach from the leader. Consequently, only one spanning tree is possible. In this case, the *Ap* list is the same as the base tree $T0$, which is the only spanning tree for the entire structure. No further variants are generated. If the configuration includes cycles, then the *Ap* list contains additional edges. This brings us to the second step, where these edges can be added to the base tree to create a temporary cycle. Then, another edge of this cycle is removed to produce a new spanning tree different from the previous one. The code explores all possible combinations of this type. Each tree is temporarily stored in a stack and then processed one by one.

After the enumeration of all valid spanning trees (Algorithm 1), the final stage consists in traversing each of these trees to perform the mechanical verification of every inter-module bond as shown in Algorithm 2. This phase aims to identify potential rupture points under the hypotheses defined by the moment-based stability criteria.

4 Experiments and Results

The experiments on Free-loop configurations were conducted on two types of platforms: the *VisibleSim* simulator and *Blinky Block* hardware. The algorithm presented has been implemented in the *VisibleSim* simulator and tested on many different robot configurations. In particular, we tested different topologies with 2- and 3-dimensional configurations with and without loops. The four types of

breaks are handled by our implementation and appear in different colors on the blocks that detected them as displayed Figure 3 (Red for vertical detachment, blue for rotational breakage and purple for lateral detachment and orange for the sliding breakage).

The algorithm has been implemented in *Blinky Block* hardware as well. The embedded program checks the stability of the configuration when powering on. A supply block is connected to the *Blinky Block* fixed on the floor, which is also the root of the spanning tree. *Blinky Blocks* that detect a breakage with their children light up in color: orange for case A, blue for case B, red for case C and purple for case D. The picture of Figure 4(a) shows an the experiment with multiple breakage detection. The supplementary video shows how the program works using several experiments with *Blinky Blocks* highlighting each of the breakage cases studied in this paper.

For Multi-Loop Configurations, the experiments were conducted using the *VisibleSim* simulator. The results presented in Figures 4(b) highlight the ability of the proposed method to comprehensively verify the stability of inter-modular connections for different types of configurations. A red leader indicates that instability was detected, while a green leader indicates no instability. Our tests applied to a large set of reconfigurations shows that the method is robust. We test it to simulated modules with a different number of connection as well.

Complexity: We can now express the complexity of this distributed algorithm. Since processing time is primarily due to message delays, time complexity is related to the number of messages transferred to perform the computation. The complexity of the first algorithm can be expressed as: $O(k_t d)$ where d is the diameter of our configuration and the height of our equilibrated spanning tree, and k_t is a constant representing the message transmission delay. Tackling loops problem, the processing complexity of a loop is $O(N^2_{\text{loop}})$. When multiple loops

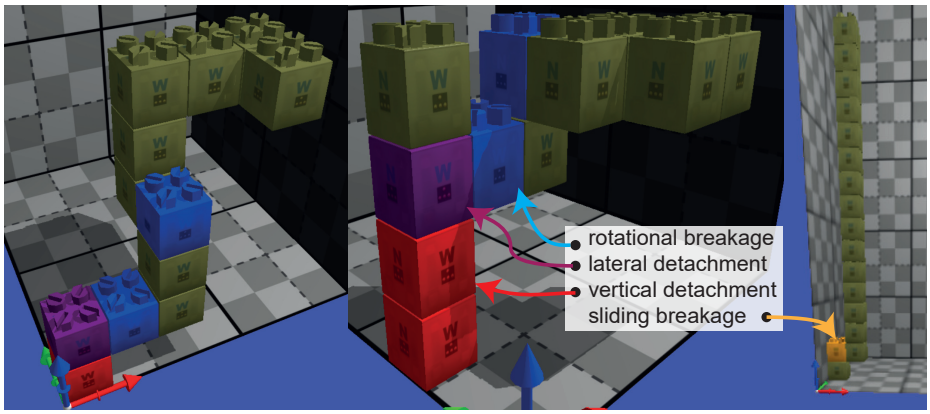
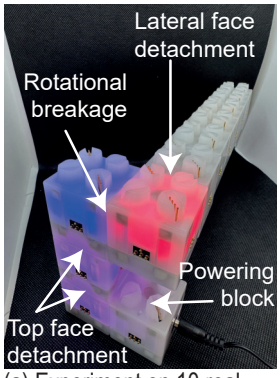
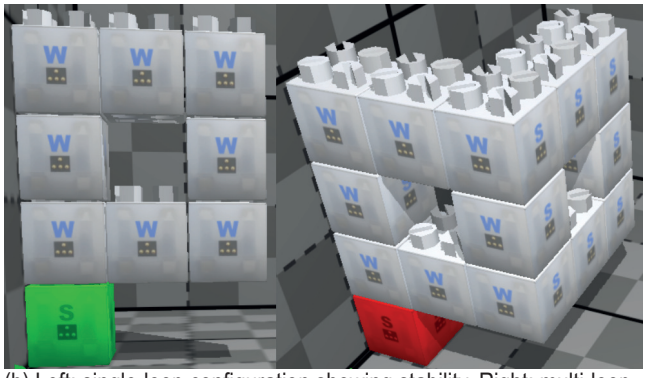


Fig. 3. Example of breakage detection, each colored block detects a kind of breakage on the face connected to the free part of the structure.



(a) Experiment on 10 real *Blinky Blocks*.



(b) Left: single-loop configuration showing stability. Right: multi-loop configuration showing instability.

Fig. 4. Several experiment on *Blinky Block* Hardware and *VisibleSim* simulator.

are present, only the cuts on the common modules require combining both module sets. If m modules are common: $O((1+m)(N_1^2 \times N_2^2))$.

5 Conclusion

The subject addressed in this article aims to deal quickly and effectively with the mechanical stability of a configuration for programmable matter.
The proposed algorithms implement hierarchical communication between the network's modules, initiated by the leader and enabling dynamic, progressive verification of system breakages. In the case of free-loop configurations, each module participates in the collection of information, progressively aggregating it and transmitting it to its parent, thus verifying the breakages likely to occur at all these interfaces. Finally, the leader gathers all the data to display the modules in which one or more breakages have been detected.

However, in the case of single or multi-loop configurations, rather than solving a global system of coupled equilibrium equations, which would require intensive message exchange and long convergence times in a distributed context, the proposed solution relied on an innovative decomposition strategy. By enumerating all possible spanning trees and systematically analyzing their stability, the algorithm effectively transformed the complex looped problem into a set of simpler, acyclic cases.

The experimental results confirmed that the proposed algorithms of both cases, accurately identified rupture interfaces, with reasonable computational cost even for large-scale assemblies.

Acknowledgments. This work has been supported by the EIPHI Graduate School and the SELF-CONTROL Project "ANR-17-EURE-0002".

References

1. Bourgeois, J., Piranda, B., Naz, A., Tucci, T., Mabed, H., Dhoutaut, D., Boillot, N., Lakhlef, H.: Programmable matter as a cyber-physical conjugation. In: International Conference on Systems, Man, and Cybernetics. Budapest, Hungary (Oct 2016). https://doi.org/10.1109/SMC.2016.7844687
2. Bray, E., Groß, R.: Distributed optimisation and deconstruction of bridges by self-assembling robots (June 2022), https://eprints.whiterose.ac.uk/187863/, © 2022.
3. Goldstein, S.C., Campbell, J.D., Mowry, T.C.: Programmable matter. IEEE Computer **38**(6), 99–101 (June 2005), http://www.cs.cmu.edu/~claytronics/papers/goldstein-computer05.pdf
4. Kirby, B., Campbell, J., Aksak, B., Pillai, P., Hoburg, J., Mowry, T., Goldstein, S.C.: Catoms: Moving robots without moving parts. In: Proceedings of the National Conference on Artificial Intelligence. vol. 20/4, p. 1730. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2005)
5. Piranda, B., Bourgeois, J.: Designing a quasi-spherical module for a huge modular robot to create programmable matter. Autonomous Robots **42**(8), 1619–1633 (dec 2018). https://doi.org/10.1007/s10514-018-9710-0, http://link.springer.com/10.1007/s10514-018-9710-0
6. Piranda, B., Bourgeois, J.: Datom: A deformable modular robot for building self-reconfigurable programmable matter. In: 15th International Symposium on Distributed Autonomous Robotic Systems (DARS 2021). Kyoto, Japan (jun 2021), https://publiweb.femto-st.fr/tntnet/entries/17404/documents/author/data
7. Piranda, B., Chodkiewicz, P., Hołobut, P., A. Bordas, S.P., Bourgeois, J., Lengiewicz, J.: Distributed prediction of unsafe reconfiguration scenarios of modular robotic programmable matter. IEEE Transactions on Robotics **37**(6), 2226–2233 (2021). https://doi.org/10.1109/TRO.2021.3074085
8. Romanishin, J., Gilpin, K., Rus, D.: M-blocks: Momentum-driven, magnetic modular robots. In: IROS. pp. 4288–4295 (2013)
9. Spröwitz, A., Laprade, P., Bonardi, S., Mayer, M., Moeckel, R., Mudry, P.A., Ijspeert, A.J.: Roombots—Towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on (Oct 2010). https://doi.org/10.1109/IROS.2010.5649504
10. Swissler, P., Rubenstein, M.: Fireant: A modular robot with full-body continuous docks. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation (2018)
11. Thalamy, P., Piranda, B., Naz, A., Bourgeois, J.: Visiblesim: A behavioral simulation framework for lattice modular robots. Robotics and Autonomous Systems p. 103913 (2021). https://doi.org/https://doi.org/10.1016/j.robot.2021.103913, https://www.sciencedirect.com/science/article/pii/S0921889021001986
12. Yazidi, I., Piranda, B., Ouisse, M., Bourgeois, J.: Balance-driven self-reconfiguration algorithm for programmable matter. In: 17th International Symposium on Distributed Autonomous Robotic Systems (DARS 2024). Springer, New-York, USA (October 2024)
13. Yazidi, I., Piranda, B., Ouisse, M., Bourgeois, J.: Efficient balance detection for modular robots. In: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 14119–14124 (2024). https://doi.org/10.1109/IROS58592.2024.10802149