Balance-driven Self-Reconfiguration Algorithm for Programmable Matter

Ikrame Yazidi, Benoît Piranda, Morvan Ouisse and Julien Bourgeois

FEMTO-ST Institute, Université de Franche-Comté, CNRS
firstname.lastname@femto-st.fr

Abstract. In this paper, we propose a fully distributed method for scheduling the movements of modular robots in a set while maintaining its balance. We explore self-reconfigurable modular robots, which are robots that can change their shape and behavior on their own. We focus on the important problem of verifying their stability and keeping these robots stable while they change shape.
Our approach involves developing an algorithm that help the robots stay balanced in real-time, even when they are changing shape. At the heart of this algorithm is the mechanical notion of the "support polygon" and the concept of "*Mobile Object* " in message data, which enables a fast stability verification. We test this algorithm using a programmable matter simulator called *VisibleSim*, we show that it works well for validating a current static situation and for predicting the status after one or more movements. Additionally, we introduce the concept of dimensional reduction from 3D to 2D, simplifying stability analysis while maintaining accuracy. By addressing stability challenges, our research aims to make self-reconfigurable modular robots more useful in various applications.
Keywords: Self-reconfiguration, Modular robots, Distributed algorithms, Stability, Programmable Matter.

1 Introduction

Self-reconfigurable modular robots represent a revolutionary advance in modern robotics [1,2,3]. These innovative robots are designed to be adaptable, flexible and capable of autonomous reconfiguration to meet a variety of situations and requirements. Their potential is immense, covering a wide range of applications, from space research and manufacturing to medicine and logistics.
The growing interest in self-reconfigurable modular robots lies in their ability to solve complex problems efficiently, and reduce the need to design specific robots for particular tasks. Indeed, these robots can adjust their behavior and hardware configuration to adapt to constantly changing environments, while optimizing their performance to achieve their objectives. This adaptability makes them particularly attractive for situations where traditional robots would be limited or ineffective. A system of this kind would be a realization of the futuristic concept of Programmable Matter [4,5].
However, despite their potential, self-reconfigurable modular robots also present

considerable challenges. One of the major challenges is the stability of these set of robots, both in terms of mechanical stability, ensuring that the robot does not accidentally come apart during its operations, and the stability of the algorithm itself, ensuring that reconfiguration processes run reliably and consistently.
The ability to reconfigure can sometimes lead to instabilities or control problems, which can compromise their usefulness in critical applications. This stability issue represents a key area of research in the development of these robots, and its resolution is essential for realizing their full potential.
The main objective is to develop distributed algorithms and control methods that are fast and extensible across a large number of modules, enabling modular robots to assess and maintain their stability in real time, based on self-control. Self-control refers to the robot's ability to verify its stability and to adjust its actions to maintain balance. This involves analyzing the various possible configurations of the modules, assessing the weight distribution, and determining the measures needed to correct any instabilities identified. Hence, the introduction of the notion of "support polygon"and "*Mobile Object* ".

The purpose of this article is then to focus on the crucial issue of stability, examining the challenges, advances and future prospects in this field and to propose solutions based on self-control to verify the stability of the set of robots, and predict stability of possible configurations reached after several motions of robots. We will also show that robot self-reconfiguration, which is a complex 3D problem, can be simplified to a 2D problem if it is driven by the mechanical balance of the set of modules.

2 Related Works

Many previous works have put the accent on the stability verification problem for self-reconfigurable modular robots, such as [6] and [7]. An approach using the Finite Element Method [8] and related spring-mass discretization has been used to study mechanical properties of modular structures in [9]. In [10] and [11], the authors investigated scalable collective actuation in a special class of modular structures. In [12], Bray and Groß used a distributed control strategy to reconfigure self-assembled structures that fills a void while incorporating local force measurements to build structures that do not colapse under their own weight. In [13], Piranda et al. propose an algorithm based on a distributed solution of mechanical equilibrium equations derived from a simplified model of the robot. However, it takes a long time to verify the balance, even for small configurations. In this paper we address the same objective as in [12] of maintaining stability while moving, but we use a different approach based on weight distribution. We have developed the approach introduced in [13], in the case where the robot is rigid and stands on a flat ground, but using a different method for verification. The method used in this specific case in [13], is to calculate the center of gravity, distribute the x and y coordinates, determine the safe angular ranges for each support module, add them up, and conclude that the structure is stable if the aggregated angular range is equivalent to a full angle, which is 360 degrees. This

method takes a long time to converge, even if we have a small number of support modules, as the calculation time depends on the number of support modules.

3 Context

In this work we address two distinct scenarios: The first involves a modular robot with improper weight distribution, similar to construct a table with one leg. Meanwhile, the second scenario concerns to a modular robot capable of changing shape, like constructing a bridge over a river starting from one bank and arriving to the other. The main difference between the two lies in the fact that the first remains in a fixed shape, while the second reconfigure itself. In the first case, the problem lies in the risk of the robot tipping over during assembly due to unbalanced weight distribution, while in the second case, the challenge is to prevent the robot from losing balance as it changes shape.
Thus, verifying the stability of modular robots, both during their movement and when they are not actively moving, is essential to ensure their stability to operate safely in real-life environments. Thorough mechanical analysis is required to identify and resolve potential stability problems before deployment, thus reducing the risk of critical hardware damage and malfunction. Before going into the details of the algorithm, we must first give a few definitions:

Definition 1. *The definition of the Mobile Object concept is proposed by Michel Raynal in [14]. It consists of defining a computer object such as a data structure that is exchanged sequentially by different modules via messages. In our case, this Mobile Object is made up of a list of the positions of modules in contact with the ground and the data used to calculate the center of mass of a sub-tree (i.e. the sum of the centers and the number of modules in the sub-tree).*

Definition 2. *The support polygon S is defined as the convex envelop (CE) of the set of vertices V of the robots in contact with the ground. In our situation, the ground is the horizontal plane with $z = 0$ only.*

$$S = CE(v_i \in V | v_i \cdot z = 0) \tag{1}$$

Definition 3. *Consider the graph $G(R, L)$, where the vertices are all the robots R and L is the list of point-to-point connections between two neighboring robots. We define a spanning tree of the undirected connected graph G by a tree included in G that connects all the vertices R.*

Lemma 1. *A configuration C is stable if and only if the projection of the gravity center G in the ground is placed inside the support polygon S.*

$$stability(C) = P(G.x, G.y, 0) \in S$$

Theorem 1. *Let $P(V, E)$ a polygon defined by its N CCW oriented vertices $V = \{v_0..v_{N-1}\}$ and edges $E = \{(v_0, v_1), (v_1, v_2) \dots (v_{n-2}, v_{n-1}), (v_{n-1}, v_0)\}$. A point M is inside a convex polygon if it is at the left of all edges.*

$$M \in P \iff \forall i \in [0..n], \left(\overrightarrow{v_i M} \times \overrightarrow{E_i}\right) \cdot z < 0 \tag{2}$$

Proof. To prove this by contradiction, considering that the polygon is CCW oriented, if M is to the right of a line passing through an edge, it is obvious that M cannot be inside the polygon.

4 Contribution

To determine the static state of a configuration, our solution is to propagate geometric information from the tree's leaves to the root in a spanning tree, enabling this module to determine whether or not the set of robots is stable. Secondly, in order to predict the stability of the system after a displacement of a module, we transfer the mechanical data to the module concerned, and locally compute a new potential stability for each of the displacements envisaged for the module.
We consider that we have previously applied a leader election algorithm to our set of robots and built a spanning tree (cf. Definition 3) from this module. Each robot R_i get its position C_i relatively to the leader, applying *Disco* algorithm presented in [15].

4.1 Process of Stability Verification

The process of checking the stability of a modular robot can vary according to its configuration and its condition. In the case of a static modular robot, meaning one that stays immobile on the ground, the verification is carried out once by the leader by verifying the Lemma 1.
At start, robots R_i at the leaves of the spanning tree create the *Mobile Object* $(\{(Sum.C, Sum.N), listBase\})$ with $listBase$ the list of modules on the ground (here $listBase$ is equal to L_i the list of vertices of R_i in contact with the ground), $Sum.C = C_i$ the local center of mass and $Sum.N = 1$ the number of modules. R_i sends these data to its parent neighbor by message. At reception at R_j the data of all children are combined to get the *Mobile Object* of the sub-tree defined by R_j and its children. Once the leader receives the required data $(Sum, listBase)$, it calculates the robot's center of gravity G and creates the support polygon S defined in Definition 2 using $listBase$.

Once the calculations are completed, the leader verifies the Lemma 1 to check the stability which requires the use of Theorem 1 to verify whether $(G.x, G.y)$ is inside S or not. If it lies within the polygon, this means the robot is stable. If not, it indicates an imbalance and therefore an instability.
It's interesting to be able to check whether the current state of the robots is stable, but in a second step, we'll show that we can also use the data collected to predict the stability of the assembly after a self-reconfiguration step. The module attempting to move needs the mechanical data and requests it from the leader via modular navigation by *Mobile Object*. Upon acquiring the *Mobile Object*, this module R_m becomes the new leader, then, It calculates the prediction of G^1 from G^0 by removing its position C_m^0 and adding the position of its potential

destination C_m^1. It is worth mentioning that the mass of the *3D Catoms* does not play any role into the calculations since all the *3D Catoms* have the same mass. Then, it verifies whether the polygon will change or not to predict the new polygon.

$$\begin{cases} G^1 = G^0 + \dfrac{C_m^1 - C_m^0}{Sum \cdot N} \\ S^1 = S^0 + \{P_i \in R_m^1 \wedge P_i.z = 0\} - \{P_i \in R_m^0 \wedge P_i.z = 0\} \end{cases} \tag{3}$$

After that, the module R_m checks whether the modular robot will remain stable in this new configuration. If so, it is authorized to proceed with its reconfiguration. If not, it cancels its movement. To simplify in this work, we consider that when moving from one stable position to another, the modular robot remains stable. It would be interesting to verify that the balance of the system for the two extreme positions leads to stability for all intermediate positions and this for the two different types of relative motion (along hexagonal and octagonal paths cf. [16]). This process repeats each time a module attempts to change its position. In this way, the stability verification process is continuous and adaptive, ensuring that the modular robot maintains its stability while undergoing configuration changes, and avoiding any risk of falling over or becoming unstable.

4.2 Fully Distributed Algorithm

The proposed algorithm is designed as a fully distributed tree-based partitioning algorithm that is based on the idea that modular robots can communicate with

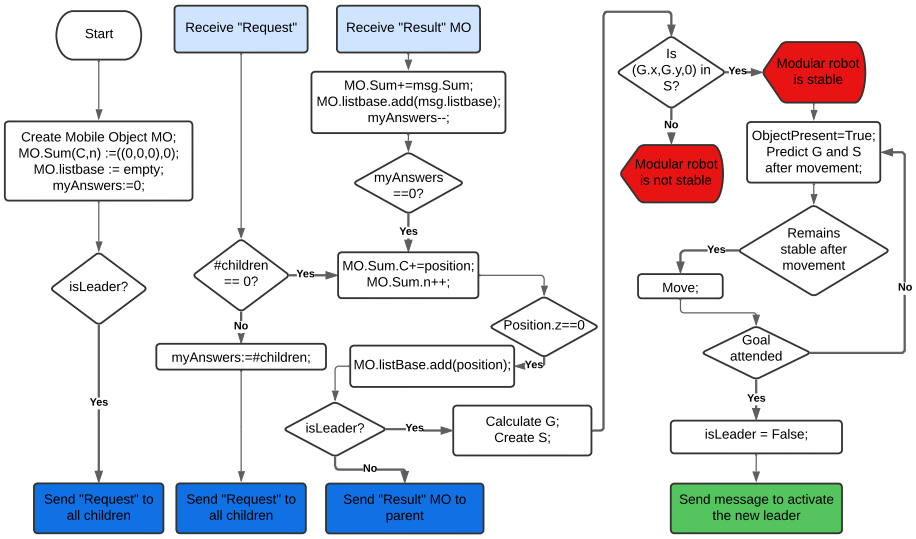


Fig. 1: Distributed Algorithm for Stability Verification.

each other, exchanging information such as their position. This communication is crucial for effective cooperation between modules. In this context, the use of a spanning tree becomes of great importance and the first thing to start with. Fig. 1 shows the first part of the distributed algorithm which is dedicated to stability verification before any movement. The sent messages are in dark colors, while the received messages are in light colors. A notable aspect of this part is that it clearly displays whether the modular robot is stable or unstable in each of the simulated scenarios. The part that follows displaying results shown in red connects Fig. 1 with Fig. 2. Moreover, the green-colored message response and its subsequent actions are illustrated in Fig. 2 that is focusing on the message exchange between modules to acquire the *Mobile Object*. At the beginning, the leader possesses the *Mobile Object* and employs it to verify stability before each movement until it attends its final position. Then, Each module wishing to initiate a movement should request the *Mobile Object* from the leader. Upon acquiring it, this module becomes the new leader, replacing the former leader.

In this section, we detail the algorithm by presenting two pseudo codes. The stability verification is presented in Algorithm 1, while the stability prediction is detailed in Algorithm 2. In Algorithm 1 the `Startup` function initializes the process sending a "request" message from the leader to all its neighbors. `myBroadcastFunc` handles incoming "request" messages where each module forwards, the request message, to all its children and if no children, transmits the desired response to its parent. `myAcknowledgeFunc` handles incoming "response" messages. When a module receives the expected response from all its children, it

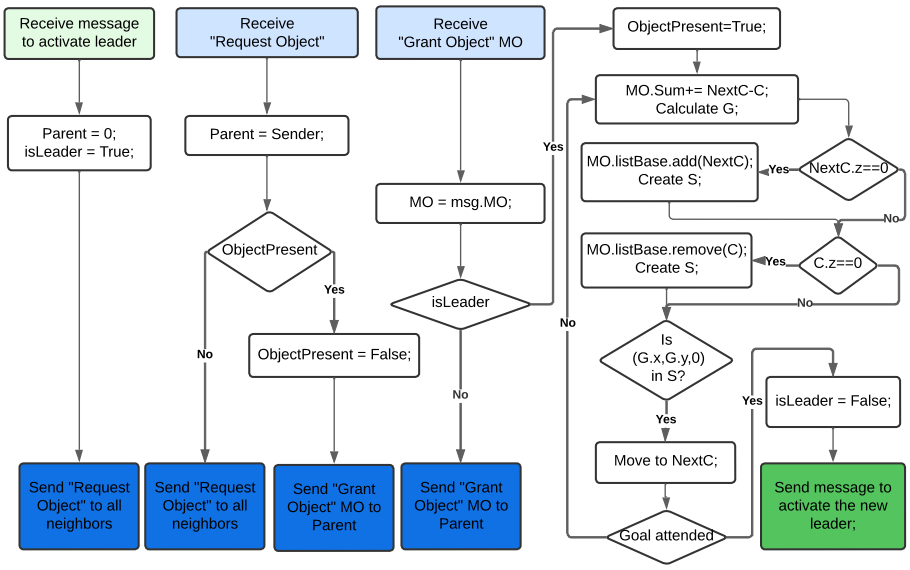


Fig. 2: Distributed Algorithm for Stability Prediction.

Algorithm 1: Stability Verification Algorithm.

```
1  Function Startup():
2      if isLeader then
3          nbWaitedAnswers ← sendMessageToAllNeighbors("Request");
4  Msg Handler myBroadcastFunc(msg, sender):
5      if parent=∅ then
6          parent ← sender;
7          nbWaitedAnswers ← sendMessageToAllNeighbors("Request");
8          if nbWaitedAnswers=0 then
9              Calculate MO.Sum and MO.listbase;
10             sendMessage("Result", MO, parent);
11     else
12         sendMessage("Result", empty, sender);
13 Msg Handler myAcknowledgeFunc(msg, sender):
14     nbWaitedAnswers−−;
15     if ¬msg.empty() then
16         Add msg.Sum to MO.Sum and Insert msg.listbase to MO.listbase;
17     if nbWaitedAnswers=0 then
18         Calculate MO.Sum and MO.listbase;
19         if parent=∅ then
20             Calculate G and S, Verify stability and display result;
21         else
22             sendMessage("Result", MO, parent);
```

Algorithm 2: Modular Navigation Algorithm by Mobile Object.

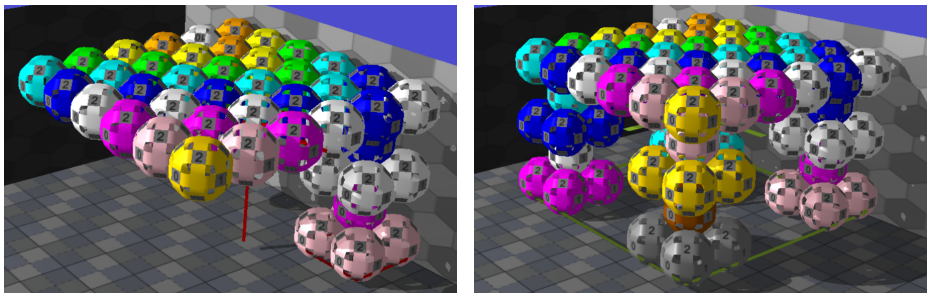
```
1  Function RequestMobileObject():
2      SendMessageToAllNeighbors("Request Object");
3  Msg Handler myBroadcastFunc2(msgMO, sender):
4      parent = sender;
5      if object_present then
6          object_present = false;
7          sendMsg("Grant Object", MO, parent);
8      else
9          SendMessageToAllNeighbors("Request Object");
10 Msg Handler onMessageReceived(msgMO, sender):
11     MO = msgMO;
12     if parent=∅ then
13         object_present = true;
14         Calculate new G and Create new S if it changes;
15         Check stability and try to move;
16     else
17         sendMessage("Grant Object", MO, parent);
```

forwards it to its parent, until the leader that calculates G and S and performs the stability verification.

In the Modular Navigation Algorithm by Mobile Object, outlined in Algorithm 2, each module seeks to coordinate its movement within the others. Once initiating a movement request and requiring the mobile object, the module broadcasts a 'Request Object' message to its neighbors. Upon receiving this message, neighboring modules either grant the requested object or continue propagating the request. Once a mobile object is obtained, the leader computes necessary parameters, checks for stability and initiates the movement accordingly. This process is repeated each time a module attempts to move.
Regarding the complexity in time of the algorithm, it is directly associated with the number of communications produced during the computation of the stability. It depends on the height of the spanning tree, which is essentially the eccentricity of the root node. This distance is majored by the diameter of the graph representing the configuration of the modules. Hence, we can conclude that the time complexity is $O(diameter)$.

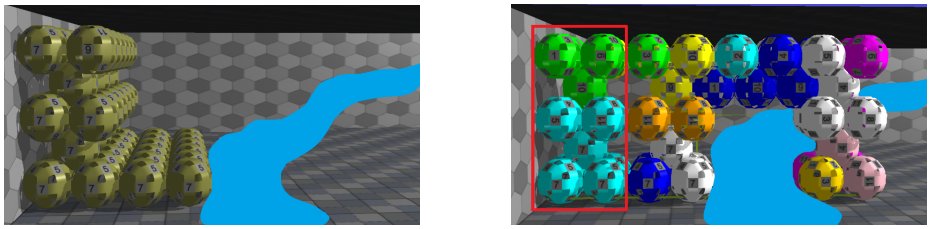
5 Experiments and Results

The proposed algorithm was implemented in *VisibleSim* [17], a framework for creating behavioral simulators for lattice-based distributed modular robotic systems in a normal 3D environment. There are many modular robots currently supported by *VisibleSim*, these include *3D Catoms* [16] which are quasi-spherical micro-robots able to stick to each other and to move around each other. *3D Catoms* are placed in a regular FCC grid and each of them can have up to 12 neighbors. Our algorithm was implemented in *VisibleSim* to simulate the behavior of modular robots and to assess their stability in different scenarios. In this experiment we draw a 3D mark at the center of mass position and the edges of the support polygon, at green if the configuration is stable and red otherwise. We have addressed the two examples outlined in section 3: A table, to study the



(a) Unstable Table with only 1 Leg. (b) Stable Table with 4 Legs.

Fig. 3: Detection of Balance of 2 Configurations, lines drawing the center of mass position and support polygon is green if stable red otherwise.



(a) Initial Shape. (b) Bridge Over a River.

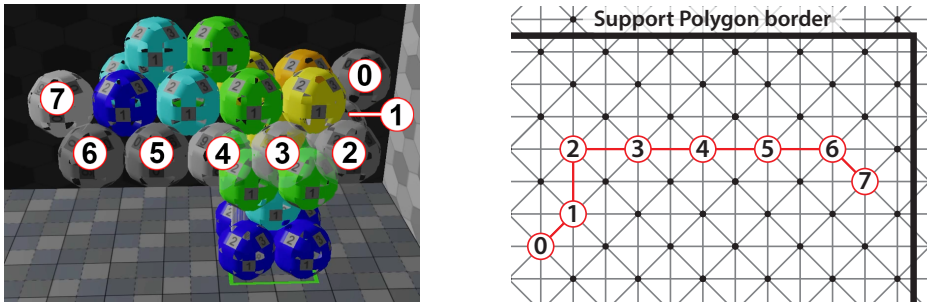
Fig. 4: Self-Reconfiguring *3D Catoms* Constructing a Bridge Over a River.

stability of a static case, and a modular robot self-reconfiguring into a bridge over a river to examine the step by step verification.

For our first example, we check in Fig. 3a that a table with only one leg is an unstable configuration, leading us to conclude that such a configuration cannot be built in reality. To address the stability problem, we added the other three legs. The simulations depicted in Fig. 3b show how adding ground modules can expand the support polygon, ensuring the robot's stability.
In the second example, we intentionally considered a river below, complicating the task since we cannot use modules on the ground to widen the support polygon, and the challenge of reaching the other side safely remains. Fig. 4a shows the initial state of the modular robot before any movement. Ultimately, we successfully constructed our bridge safely, as shown in Fig. 4b, by using another approach to address the stability issue, which is adding counterweights during reconfiguration. The addition of these counterweights can adjust the position of the gravity center, thus helping to stabilize the modular robot. The red-highlighted section in Fig. 4b represents the modules that acted as counterweights and which can be removed once we have achieved our goal.

6 Discussion

As part of our approach, we will be taking a close look at the concept of step-by-step stability verification. One of the key aspects we will delve into is 3D to 2D Dimensional Reduction. Consider the 3D representation of the robot in Fig. 5a, where the white module is drawn at its successive predefined positions during reconfiguration. Throughout this journey, there is a risk of the modular robot losing its balance. To get a clearer view of how we ensure stability, we switch from a 3D to a 2D view, by ignoring the third dimension (z-axis). This simplification is possible because we only need to focus on two factors, G_x and G_y, along with the support polygon, to check for stability. This transition to a 2D representation enables us to visualize more clearly the changes occurring during the module's movement.



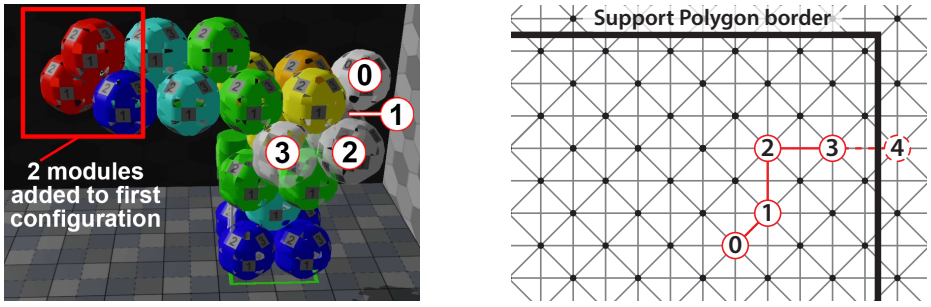
(a) 3D Modular Robot Trajectory. (b) *GC* Projection Trajectory.

Fig. 5: Self-Reconfiguration Process.

Clearly, the center of gravity G moves with each module movement, as shown in Fig. 5b. As observed, the projection of the center of gravity remains within the support polygon throughout the movements. Consequently, the module successfully reaches its final position.

Let us consider another scenario where we add two modules as shown in Fig. 6a. In this new situation, the 4th displacement of the white module causes a loss of stability for the set. The mobile module can anticipate this problem and abort the movement. For a better comprehension, let us examine the 2D representation in Fig. 6b that shows that in the case where the module moves, the center of gravity, shown in red, will no longer be inside the support polygon. The last position that the module can reach while maintaining the stability of the modular robot is the position of the white module shown in Fig. 6a. The movements of the white module in both scenarios presented in this section are clearly shown in the video[1] it shows in simulation the several steps of the self-

[1] Video link: https://youtu.be/sg5ef-wanKo



(a) Safe positions for 2^{nd} configuration. (b) Unsafe Position Detection.

Fig. 6: Detection of Unsafe Motion.

reconfiguration into a bridge too, displaying the addition of counterweight when it is needed, while clearly illustrating the changes in center of gravity positions and support polygon with each movement. According to Fig. 5b and 6b, we notice that the center of gravity moves in translation by 2 distances, either ΔG_1 or ΔG_2, between each positions. Thus, let us demonstrate where ΔG_1 and ΔG_2 come from and how we can determine them.
Upon analyzing the movements of the *3D Catoms*, we observe that each module can only perform a maximum of 8 displacements. Consequently, we concluded that the possible positions to which a module (x, y) can move are summarized in : $(x \pm 1, y)$, $(x, y \pm 1)$ and $(x \pm \frac{1}{2}, y \pm \frac{1}{2})$.
After determining the possible positions for each movement of the *3D Catoms* and since the number of modules N remains constant, we have all the parameters necessary to predict the possible values of the center of gravity after each displacement. In our analysis, we will focus only on (G_x, G_y). Considering G^0 the center of gravity of the modular robot before movement and G^1 the center of gravity of the modular robot after the displacement of any module.

$$G_x^0 = \frac{\sum x_i}{N} \quad \text{and} \quad G_y^0 = \frac{\sum y_i}{N}$$

Let us proceed and calculate the possible values of (G_x^1, G_y^1):

(a) The first possibility where the next position is $(x \pm 1, y)$:

$$\begin{cases} G_x^1 = \dfrac{\sum x_i - x + x \pm 1}{N} = G_x^0 \pm \dfrac{1}{N} \\ G_y^1 = \dfrac{\sum y_i}{N} = G_y^0 \end{cases} \tag{4}$$

(b) The second possibility where the next position is $(x, y \pm 1)$, we obtain: $G_x^1 = G_x^0$ and $G_y^1 = G_y^0 \pm \dfrac{1}{N}$.

(c) The Third possibility where the next position is $\left(x \pm \frac{1}{2}, y \pm \frac{1}{2}\right)$, so we get: $G_x^1 = G_x^0 \pm \dfrac{1}{2N}$ and $G_y^1 = G_y^0 \pm \dfrac{1}{2N}$.

As a consequence, after any motion the center of gravity can move into 8 different positions only. Therefore, we can generalize and say that, with each movement of a module, the center of gravity will have one of the predefined values. According to these calculations, we can express the variation of the projection of the center of gravity as $\Delta G_1 = \frac{d}{N}$ or $\Delta G_2 = \frac{d}{\sqrt{2}N}$ where d is the diameter of the *3D Catom*. Knowing this distance is crucial since it will allow us to determine if the projection of the center of gravity will remain within the support polygon or not. Thus, it will open the way for a new method of stability prediction.
According to all that precedes, we can thus create a grid that represents all the possible positions of the center of gravity while respecting the possible distances ΔG_1 and ΔG_2. The Fig. 7 shows how our proposed grid is made. Even and Odd grid are $\frac{d}{N}$ large and interleaves to create the list of possible positions for the projection of the center of mass. We have shown that the process of

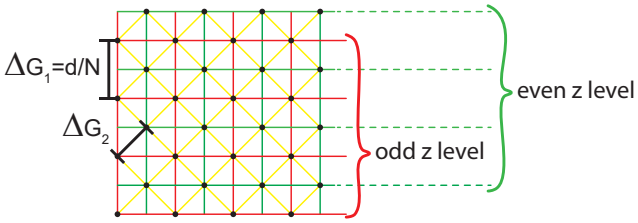


Fig. 7: Grid of possible positions of the center-of-mass projection on the ground.

verifying stability after one displacement is $O(1)$ complex, which is advantageous compared to the complete recalculation of stability ($O(diameter)$ complex), since the diameter can be important in the case of programmable matter.

7 Conclusion and Future Work

In this article, we delved into the world of self-reconfigurable modular robots, focusing particularly on the crucial issue of stability. Throughout our exploration we investigated various stability verification methods and proposed a new approach based on the creation of support polygons and modular navigation by mobile object using distributed algorithms.
In our research, we developed a simulation model to study the behavior of modular robots and assess their stability in different scenarios. Through experiments and results analysis, we show the effectiveness of our approach in ensuring stability, both in static configurations and in predicting status after movements. Furthermore, we explored the concept of dimensional reduction from 3D to 2D to simplify stability analysis.
We have proposed a new method for rapid stability prediction, which qualifies as stable or not the various displacements of the center of gravity according to the possible movements. Looking ahead, our research opens up other avenues for future work in the field of self-reconfigurable modular robots. One promising direction is the development of more advanced stability algorithms, incorporating multiple movements where several *3D Catoms* move simultaneously, Combined movements meaning one movement balancing another and continuous trajectory calculation between two positions.
In conclusion, our research contributes to advancing the field of self-reconfigurable modular robots by addressing critical stability challenges and proposing innovative solutions. By further exploring the avenues outlined in this article, we can continue to enhance the stability, adaptability, and efficiency of modular robots for a wide range of practical applications.

Acknowledgement

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

References

1. A. Kamimura, S. Murata, E. Yoshida, H. Kurokawa, K. Tomita, and S. Kokaji, "Research on self-reconfigurable modular robot system: (experiments on reconfiguration and locomotion with several modules)," *Nippon Kikai Gakkai Ronbunshu, C Hen/Transactions of the Japan Society of Mechanical Engineers, Part C*, vol. 68, no. 3, pp. 886–892, Mar. 2002.
2. M. Yim, W.-m. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *Robotics Automation Magazine, IEEE*, vol. 14, pp. 43 – 52, 04 2007.
3. J. Lengiewicz, M. Kursa, and P. Hołobut, "Modular-robotic structures for scalable collective actuation," *Robotica*, vol. 35, no. 4, p. 787–808, 2017.
4. S. C. Goldstein, J. D. Campbell, and T. C. Mowry, "Programmable matter," *IEEE Computer*, vol. 38, no. 6, pp. 99–101, June 2005. [Online]. Available: http://www.cs.cmu.edu/~claytronics/papers/goldstein-computer05.pdf
5. J. Bourgeois, B. Piranda, A. Naz, T. Tucci, H. Mabed, D. Dhoutaut, N. Boillot, and H. Lakhlef, "Programmable matter as a cyber-physical conjugation," in *International Conference on Systems, Man, and Cybernetics*, Budapest, Hungary, Oct. 2016.
6. P. Holobut and J. Lengiewicz, "Distributed computation of forces in modular-robotic ensembles as part of reconfiguration planning," in *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. IEEE, 2017, pp. 2103–2109. [Online]. Available: https://doi.org/10.1109/ICRA.2017.7989242
7. P. Holobut, S. P. A. Bordas, and J. Lengiewicz, "Autonomous model-based assessment of mechanical failures of reconfigurable modular robots with a conjugate gradient solver," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*. IEEE, 2020, pp. 11 696–11 702. [Online]. Available: https://doi.org/10.1109/IROS45743.2020.9341232
8. S. Zhang and E. Fasse, "A finite-element-based method to determine the spatial stiffness properties of a notch hinge," *Journal of Mechanical Design - J MECH DESIGN*, vol. 123, 03 2001.
9. P. J. White, S. Revzen, C. E. Thorne, and M. Yim, "A general stiffness model for programmable matter and modular robotic structures," *Robotica*, vol. 29, no. Special Issue 01, pp. 103–121, 2011.
10. P. Holobut, M. Kursa, and J. Lengiewicz, "A class of microstructures for scalable collective actuation of programmable matter," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2014, Chicago, IL, USA, September 14-18, 2014*. IEEE, 2014, pp. 3919–3925. [Online]. Available: https://doi.org/10.1109/IROS.2014.6943113
11. J. Lengiewicz, M. Kursa, and P. Holobut, "Modular-robotic structures for scalable collective actuation," *Robotica*, vol. 35, no. 4, pp. 787–808, 2017. [Online]. Available: https://doi.org/10.1017/S026357471500082X
12. E. Bray and R. Groß, "Distributed optimisation and deconstruction of bridges by self-assembling robots," June 2022, © 2022. [Online]. Available: https://eprints.whiterose.ac.uk/187863/
13. 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.

14. M. Raynal, *Distributed algorithms for message-passing systems.* Springer, 2013, vol. 500.
15. B. Piranda, F. Lassabe, and J. Bourgeois, "Disco: A multiagent 3d coordinate system for lattice based modular self-reconfigurable robots," in *IEEE International Conference on Robotics and Automation (ICRA 2023)*, London, England, may 2023.
16. 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: http://link.springer.com/10.1007/s10514-018-9710-0
17. 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