Efficient Balance Detection for Modular Robots

Ikrame Yazidi, Benoît Piranda, Morvan Ouisse and Julien Bourgeois

***Abstract*— In this paper, we explore the field of self-reconfigurable modular robots, representing a significant advance in robotic technology. These robots have many capabilities, offering high adaptability and flexibility for a variety of applications. However, computing the stability is challenging as it is computationally intensive, it needs to be distributed and fast, as close as possible of real-time. In this article, we introduce a distributed algorithm designed to overcome these challenges while taking mechanical constraints into account. At the heart of this algorithm is the notion of the "support polygon", which enables the stability of a modular robot to be assessed in real time. The algorithm is based on a fully distributed tree partitioning approach, facilitating efficient communication and collaboration between modules. The algorithm also uses a polygon merging approach to reduce the number of messages when creating the polygon support, thus significantly reducing response time. In fact, the response time of the method used is very small compared to other research. We also present simulation results on a simulator, *VisibleSim*, as well as experimental validation on real robotic modules, which underlines the practical viability of the approach. Overall, this work lays a solid base for further advances aiming to guarantee the stability of modular robots.**

Keywords: Modular robots, Distributed algorithms, Stability, Support polygon.

I. Introduction

Programmable matter [1], [2] can change its physical properties, such as color and even shape, by reconfiguring itself. This revolutionary capability opens the way to innovative applications and a futuristic vision of robotics and materials technology. Programmable matter can be implemented using self-reconfiguring modular robots [3], [4], [5], arousing growing interest among engineers and researchers. Their innovative design aims to offer exceptional adaptability and flexibility, enabling autonomous reconfiguration to meet a variety of requirements. These systems open up new perspectives in a wide range of application fields.

However, despite their promising potential, these robots present complex challenges. Stability remains a major concern, both from a mechanical and algorithmic point of views, requiring in-depth resolution to exploit their full potential. The danger lies in the fact that their design can influence their ability to maintain a solid and coherent structure, which could have potentially damaging consequences and compromise their usefulness in critical applications. If a modular robot is autonomous, its stability must therefore be collectively verified and controlled by the modules, taking into account mechanical constraints. Mechanical constraints result from the need to ensure the integrity and stability of the entire modular robot, so that the structure cannot break or lose its balance during deployment in real-life scenarios.

Currently, the majority of algorithms for planning and controlling the self-reconfiguration of modular robots do not take mechanical constraints into account. Only few of them take these constraints into consideration. An approach using the finite element method (FEM) was presented in [6] and [7] and an associated mass-spring discretization was used to study the mechanical properties of modular structures in [8], while constraints in a special class of modular structures were also examined in [9] and [5]. In [10], 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 takes mechanical constraints into account remains an open and complex problem, which is tackled in this article.

Our main objective is to develop a rapid decision tool for interactive distributed planning which takes mechanical constraints into account and is capable of verifying the stability of modular robots through the development of distributed algorithms. The ability for a modular robot to verify its own stability is defined as self-control.

To illustrate the effectiveness of our approach, we will present a large-scale demonstration using different numbers of modules.

We address the same objective as in [10] of stability verification, but we use a different approach based on weight distribution allowing a faster answer. We have developed the approach introduced in [7], in the case where the robot is rigid and stands on a flat ground. The approach relies on maintaining the center of gravity above the convex envelope of support points, but we used a completely different method for stability verification. Indeed, in this paper, the stability verification procedure is based on the creation of the support polygon using a fusion of elementary polygons, which considerably reduces the number of modules to be sent, retaining only those needed to determine the convex envelope. Our method is applicable to all types of ground contact, unlike the one described in [7], which is fast but less general. In [7] they also used another approach based on the finite element method, which is more accurate but more time-consuming. Our results are fast, general and suitable for self-reconfiguration contexts. The verification is carried out in a dedicated simulator *VisibleSim* [11], as well as experimentally on real robotic modules *Blinky Blocks* [12].

II. Exploring Modular Robot Stability

With its versatility and adaptability, the *Blinky Blocks* [12] prove to be a fitting choice for approaching the challenge of

stability verification in the context of modular robots.

The interconnected system of *Blinky Blocks* constitutes a distributed system with the following characteristics:

- All modules share a common coordinate system, and each module stores its coordinates locally, as described in [13], [14]. Modules placed on the ground will have a z-coordinate equal to 0.
- A *Blinky Block* can react to the reception of a message and the connection or disconnection of a neighbor.
- All *Blinky Blocks* run the same distributed program, performing calculations locally within each module.
- Communication occurs in a localized manner, allowing a module to exchange messages exclusively with its directly connected neighbors.
- The interconnection graph must maintain continuous connectivity.

Stability verification in the context of modular robots is crucial to ensure safe and efficient operations. An essential verification criterion in this field is weight distribution. Weight distribution refers to the balanced distribution of modules to ensure that the modular robot maintains its equilibrium under various operating conditions.

The choice of weight distribution as a stability verification criterion is based on several key factors. Firstly, it is closely linked to robot mechanics, as poor weight distribution can lead to physical instability. In addition, weight distribution can be adapted to the specific tasks the robot is intended to perform. Furthermore, the weight distribution criterion can be modified over time as the modular robot undergoes modifications or evolves to meet new needs, making it a versatile criterion for ensuring stability in a wide range of scenarios.

Checking the weight distribution generally involves several steps. First of all, a weight measurement is performed, in our case all modules have the same weight. Next, we calculate the center of gravity, usually expressed in three-dimensional coordinates. Another important step is to determine the support polygon, which is the convex envelope encompassing all points of contact between the modular robot and its support. We consider that the modular robots are placed on flat surface known as the ground. Finally, stability is verified.

The stability condition is formulated as follows: An immobile body subjected only to its own weight and the reaction of the ground is in equilibrium if, and only if, the line of action of the weight (the vertical line passing through the center of gravity) intersects the supporting surface.

In conclusion, weight distribution is a fundamental criterion for assessing the stability of modular robots, because of its direct impact on the robot's ability to maintain its balance and avoid potential falls. Therefore, this criterion is essential for ensuring the stability of modular robots. This refers to the way in which the robot's total weight is distributed between its modules.

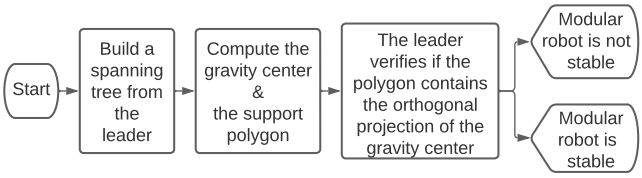


Fig. 1: Overview of the Algorithm.

III. Distributed Algorithm

A. Overview Algorithm

The proposed algorithm is designed as a fully distributed tree-based partitioning algorithm that can efficiently handle a substantial number of modules. It provides scalability by efficiently handling communication limited to neighbor-to-neighbor which means that modular robots interact and collaborate autonomously to solve the stability problem.
The algorithm is based on the idea that modular robots can communicate with 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, as shown in Figure 1. The spanning tree organizes communication between modules in a structured way, creating a hierarchy in the transmission of information, and removing loops. Its construction begins with the selection of the central node, acting as the root (leader). The selection is either manual, as we have chosen, or automated [15]. The leader connects all modules and allows to transfer information throughout the entire system.

A notable aspect of this algorithm is that it clearly displays whether the modular robot is stable or unstable in each of the simulated scenarios (See Figure 1). This provides an immediate view of the modular robot's performance under various conditions. The results of these simulations enable informed decisions to be made on the design, improvement and use of modular robots, while minimizing potential risks and optimizing their operation.

B. Detailed View Algorithm

At the heart of our method, we find a process orchestrated by a designated leader. The algorithm deploys a strategy of diffusion of relative coordinates from the leader, by flooding information through a spanning tree. As shown in Figure 2, each module actively participates by performing complex calculations.

To compute the center of mass we apply the following recursive relation at each module M of the spinning tree:

$$Sum_M(S, N) = (P_M, 1) + \sum_{\text{children } C} Sum_C \qquad (1)$$

where P_M is the position of M.

First, each module calculates Sum, which represents the sum of its relative position in the spanning tree and the coordinates of the sub-trees. Next, it determines $Poly$ by merging

its coordinates with those of its peers if its z coordinate is equal to 0. Finally, the module transmits the results to its parent, packing Sum, N, the number of modules beneath it, and $Poly$ in a message.

At the end of the algorithm, as shown in Figure 2, the leader performs global calculations to assess the system's stability. He determines GC, the ratio of the sum of coordinates to the total number of modules which is the gravity center, and $Poly$, the global support polygon:

$$GC = \frac{1}{N} \times Sum_{leader} \tag{2}$$

The robot equilibrium is established by checking whether GC is contained within $Poly$, i.e. whether the vertical line passing through the center of gravity intersects the support polygon or not.

C. Pseudo Code Algorithm

The algorithm is designed to verify the stability of modular robots, it starts with the $startup$ function as shown in Algorithm 1. If a module is designated as the leader, it sends a request to all its neighbors and waits for their replies. When a message is received, in $myBroadcastFunc$, each module updates its parent and sends requests to its neighbors if it has not already received one. It then waits for the replies and aggregates the results. When responses from all neighbors are received, in $myAcknowledgeFunc$, the module adds its coordinates to the sum and merges its polygon with those of its neighbors. If the module is the root which means the leader, it calculates the center of gravity and checks if it is inside the support polygon. Finally, it updates the color accordingly. If the module is not the root, it sends the aggregated result to its parent. In this way, the algorithm verifies the stability of the distributed system by analyzing the configuration and relative position of the modules.

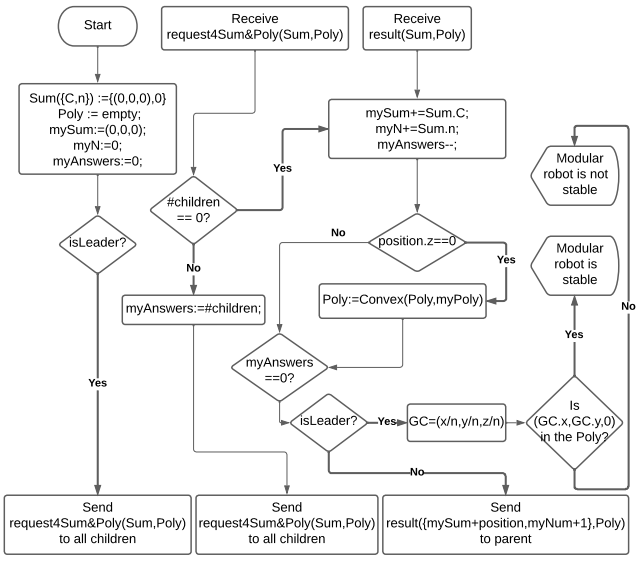


Fig. 2: Detailed View of the Algorithm.

Algorithm 1: Stability Verification Algorithm.

```
1  void startup():
2      if isLeader then
3          nbWaitedAnswers =
             sendMessageToAllNeighbors("Request")
4  void myBroadcastFunc(msg, sender):
5      if parent == nullptr then
6          parent ← sender
7          nbWaitedAnswers =
             sendMessageToAllNeighbors("Request")
8          if nbWaitedAnswers == 0 then
9              sum ← module→position
10             if module→position.pt[2] == 0 then
11                 Add the 4 vertices to Poly
12                 setColor(BLUE)
13             sendMessage("Result",(sum, Poly),parent)
14     else
15         sendMessage("Result", empty, sender)
16 void myAcknowledgeFunc(msg, sender):
17     nbWaitedAnswers−−
18     if not msg.empty() then
19         sum + = msg.first
20         Poly.merge(Poly,msg.second)
21     if nbWaitedAnswers == 0 then
22         sum + = module→position
23         if module→position.pt[2] == 0 then
24             Add the 4 vertices to Poly
25             Poly.merge(Poly,msg.second)
26             setColor(BLUE)
27         if parent == nullptr then
28             Calculate gravity center
29             if Poly.isInside((G_x, G_y)) == 1 then
30                 setColor(GREEN)
31             else
32                 setColor(RED)
33         else
34             sendMessage("Result",(sum, Poly),parent)
```

IV. Creation of the Support Polygon

The support polygon is created during communication between modules. The leader begins by sending a request to all its neighbors, and this request is then propagated. Each module that receives the leader's request sends it to its own neighbors, if it has any. This broadcasts the request from module to module, until there are no more neighbors to contact.

When a module receives the request, it checks its own position. If it is on the ground, which means, its z coordinate is equal to 0, it creates a polygon with the four sides of its perimeter, as shown for example on modules 7 and 10 in Figure 3. Once the polygon has been created (or an empty polygon if it is not on the ground), the module sends its response to its parent, meaning, to the module that sent it the request. This response contains either the polygon created

or an empty polygon.

Each module, once it has received responses from all its children (neighboring modules), merges the polygons it has received. Taking module 6 as an example, it merges the 2 polygons it received from modules 7 and 8 in Figure 3. If the module is on the ground $z = 0$, it inserts its own four sides into the global polygon resulting from the merge. This enlarges the polygon to include the contours of all ground-based modules.

Once the global polygon has been updated, each module sends the resulting polygon to its parent, until finally, the leader receives the polygon built by all the other modules. The leader also inserts its own four sides, if it is on the ground, into the global polygon, as in Figure 3 with the module 1. The final result is a support polygon drawn in red in Figure 4. The polygon contains the elements of the modular robot base, in other words, the modules that are on the ground $z = 0$.

a) Advantages of Polygon Fusion: The process of creating the support polygon is characterized by the use of polygon fusion when transmitting messages between modules. This approach offers a significant advantage over the alternative method of sending all the basic elements directly to the leader, and then building the polygon. In the polygon fusion process, instead of sending all the base elements (represented by coordinates) to the leader to create the polygon, each module transmits a polygon by merging all the polygons it has received. This method considerably reduces the amount of data to be transmitted, as only a limited set of coordinates is sent. In comparison, in an approach where all the elements of the base are sent directly to the leader, the number of coordinates to be transmitted would be proportional to the number of modules on the ground, that is, $4 \times N_0$ where N_0 is the number of modules on the ground, as shown in Figure 5a. This would result in a much more intensive transmission of data, which would be inefficient and would consume more memory, especially when we have a large number of modules on the ground.

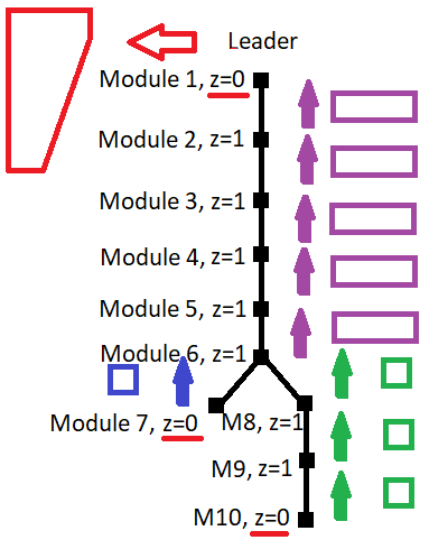


Fig. 3: Polygon Creation using Fusion.

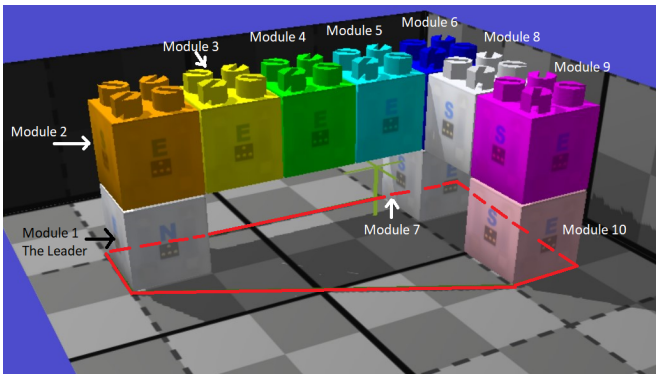
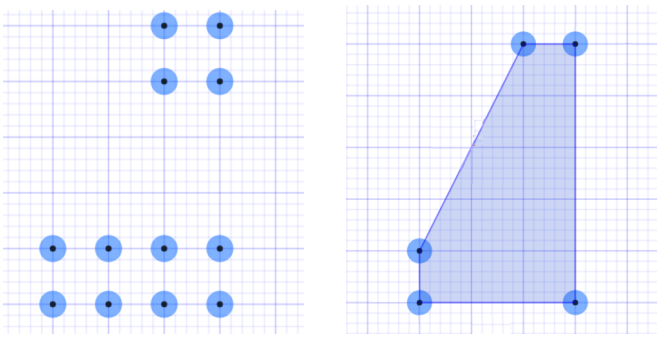


Fig. 4: Configuration used to Explain our Algorithm, with the Support Polygon in Red and Center of Mass in Green.

One of the main advantages of polygon fusion is that it automatically eliminates points located inside the convex polygon. In other words, only the points required to define the convex envelope of the base elements are retained. This ensures that the resulting support polygon effectively contains all the ground contact areas of the modules, without including redundant points within this envelope. As a result, the final polygon is more compact and more accurate, while saving unnecessary communications. As shown in Figure 5b we have reduced the number of coordinates to be sent from 12 to 5. In addition, by reducing the number of points to transmit, the size of the messages decreases. Consequently, the transmission time of the message is reduced since it is calculated based on the message size [16]. Thus, the optimization of communication through the polygon fusion process, not only minimizes unnecessary data transmission, but also efficiently reduces message size, thereby decreasing transmission time.

b) Complexity: The complexity in time of the algorithm is directly linked to the number of communications produced during the computation of the stability. It depends on the height of the spanning tree, i.e. the eccentricity of the root



(a) Coordinates without Fusion. (b) Coordinates with Fusion.

Fig. 5: Number of Coordinates to Send.

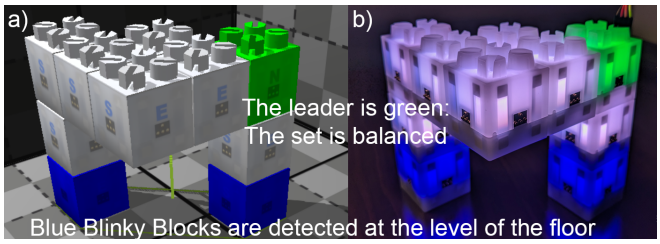


Fig. 6: The same configuration simulated on *VisibleSim* and computed by real *Blinky Blocks*.

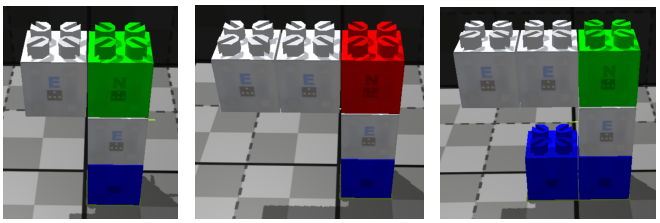


Fig. 7: Three Experiments on *VisibleSim*, the Head Block is Green if the Structure is Balanced and Red Otherwise.

node. This distance is majored by the diameter of the graph describing the configuration of the modules. We can conclude that the complexity in time is $O(diameter)$.

V. Experiments and Results

A. VisibleSim Simulation

Our development process was based on the *VisibleSim* simulator to create a robust decision-making tool. The proposed tool is used to check the stability of modular robots in various virtual scenarios. The resulting model provides a clear visualization, indicating whether the modular robot is in a stable or unstable state. For communication simulation, we used the model proposed in [16].

The Figure 6.a provides a tangible illustration of this capability, featuring our configuration with *Blinky Blocks* on *VisibleSim*. In this simulation, the modular robot reacts dynamically to the configuration of the blocks. Detection of a balanced arrangement leads to a change in the leader's color to green, signaling stability. Blue blocks detected on the floor provide additional information on the layout of the elements. The response time of this configuration was $64.326\,\mathrm{ms}$. Comparing the three illustrations in Figure 7, we see that the second one represents an unstable configuration, as the leader's color is red, while the other two illustrate stable configurations. The response time of these three configurations is $21.438\,\mathrm{ms}$, $21.447\,\mathrm{ms}$, and $32.172\,\mathrm{ms}$, respectively. We can observe significantly smaller response times, which allows to make thousands of self-reconfiguration experiments.

To overcome instability, our approach includes the use of counterweights which have the ability to modify the robot's center of gravity, thus helping to maintain its equilibrium. Another strategy is to widen the contact area with the ground, thus widening the support polygon, by adding a module to the ground, as shown in the third illustration in Figure 7. Both approaches proved effective in ensuring the stability of the modular robot in a variety of configurations.

In short, our decision tool, developed on *VisibleSim*, is proving to be a powerful resource for analyzing and improving the stability of modular robots in a variety of situations. The results obtained are opening the way to concrete applications of these modular robots in variable and complex environments.

B. Demonstration on Real Robotic Modules

In this section, we are demonstrating our decision tool on a real modular robot. To do this, we will present several experiments, each of them providing unique perspectives on the functionalities of these modules.

We have tested the same experimental configurations in simulation and in the real world, enabling a direct comparison between the theoretical results obtained on *VisibleSim* and practical observations.

Initiating this exploration, we delve into the first illustration. The Figure 6.b shows an example of how the *Blinky Blocks* configuration works and shows that the results obtained in reality are similar to the ones previously generated by the simulator. The modular robot is able to detect that the blocks are arranged in a balanced way. The leader changes its color to green to signal the stability of the configuration, while the blue blocks indicates that they are detected at the ground level. This representation reveals how the modular robot detects the balance of the blocks, signalling the stability of the configuration through color changes.

The second illustration, Figure 8 shows our three distinct configurations already used on *VisibleSim*. In the middle configuration, the leader changes its color to red, indicating that the modular robot is not stable. By removing the block responsible for the imbalance, the center of gravity is modified, resulting in a color change of the leader to green, indicating that the configuration is now stable. By retaining the initially unstable configuration and adding a ground module to widen the support polygon, the modular robot achieves stability, which is indicated by the leader changing its color to green. These examples illustrate the system's ability to react dynamically to configuration changes, showing the flexibility and adaptability of modular robots in practical situations.

After this in-depth analysis and comparison, we will move on to other example cases, providing a comprehensive understanding of the performance and practical applications of these robotic modules. The illustrations in Figures 9 highlight more complex configurations, involving larger number of modules, allowing us to visualize the remarkable performance of our decision tool. In these scenarios, modules interact synchronously to maintain balance in more elaborate structures. The effectiveness of our tool is revealed through its ability to make fast and precise decisions to ensure the stability of the whole. For example, in the second illustration in Figure 9, the leader shows that the configuration is not stable, but by adding more modules the leader changes its color to indicate that it has become stable, as shown in the third illustration in Figure 9. These examples demonstrate the

power of our decision-making system, capable of managing complex modular robot assemblies quickly and efficiently.

A video[1] completes these examples by presenting the algorithm's behavior on several configurations of *Blinky Blocks*. The implementation takes account of changes in the number of *Blinky Blocks* by restarting the calculation from the leader each time a block is added or removed.

VI. Conclusion and future work

In this article we have presented a distributed algorithm that can efficiently evaluate and guarantee the stability of modular robots. The introduction of the notion of the support polygon has proven to be essential to this task, making it possible to determine whether a robot is stable by verifying that its center of gravity remains within this convex envelope. Thus, the use of polygon fusion during communication between modules considerably reduced the data transmission load, while maintaining the accuracy of the support polygon.

We also presented the results of our simulations, using different configurations and various numbers of modules, which illustrated the benefits of the algorithm's speed feature. Experimental validation of our approach with real modular robots has also been conducted, which is a crucial step in assessing its robustness under practical conditions.

Furthermore, this work opens the way to many future research opportunities. Firstly, the optimization of the algorithm and the refinement of the stability criterion by taking into account other factors such as the forces applied to the connections and environmental characteristics remain promising areas.

Finally, an interesting possible approach is to use the minimum distance between the robot's center of gravity and the support polygon to determine the safest position for a module, thus contributing to proactive stability management. All in all, our work provides a solid basis for future investigations aimed at perfecting and exploiting this innovative approach to stability assurance for modular robots.

[1]Our algorithm in action, YouTube video: https://youtu.be/ZjUnI9osFR4

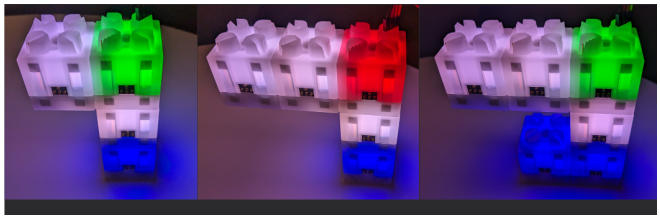


Fig. 8: *Blinky Blocks* Stable and Non Stable Configurations.

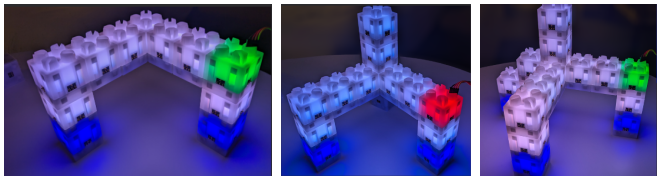


Fig. 9: Other experiments on real *Blinky Blocks*.

VII. Acknowledgement

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

References

[1] S. C. Goldstein, J. D. Campbell, and T. C. Mowry, "Programmable matter," *IEEE Computer*, vol. 38, no. 6, pp. 99–101, June 2005.

[2] 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.

[3] 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.

[4] 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.

[5] J. Lengiewicz, M. Kursa, and P. Holobut, "Modular-robotic structures for scalable collective actuation," *Robotica*, vol. 35, no. 4, pp. 787–808, 2017.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] E. Bray and R. Groß, "Distributed optimisation and deconstruction of bridges by self-assembling robots," June 2022, © 2022.

[11] 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.

[12] B. T. Kirby, M. Ashley-Rollman, and S. C. Goldstein, "Blinky blocks: a physical ensemble programming platform," in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '11. New York, NY, USA: ACM, 2011, pp. 1111–1116.

[13] B. Piranda, F. Lassabe, and J. Bourgeois, "DisCo: A Multiagent 3D Coordinate System for Lattice Based Modular Self-Reconfigurable Robots," in *International Conference on Robotics and Automation*, London, United Kingdom, May 2023.

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

[15] A. Naz, B. Piranda, S. C. Goldstein, and J. Bourgeois, "Approximate-Centroid Election In Large-Scale Distributed Embedded Systems," in *International Conference on Advanced Information Networking and Applications*, Crans Montana, Switzerland, 2016.

[16] J.-P. Yaacoub, B. Piranda, F. Lassabe, and H. Noura, "Efficient Communication Protocol for Programmable Matter," in *38th IEEE International Conference on Advanced Information Networking and Applications (AINA 2024)*, 2016.