Distributed prediction of unsafe reconfiguration scenarios of modular-robotic Programmable Matter

Benoît Piranda[1], Paweł Chodkiewicz[2], Paweł Hołobut[3], Stéphane P.A. Bordas[4], Julien Bourgeois[1], and Jakub Lengiewicz[3,4]

[1]University of Bourgogne Franche-Comté, FEMTO-ST Institute, CNRS, France
[2]Faculty of Automotive and Construction Machinery Engineering, Warsaw University of Technology, Poland
[3]Institute of Fundamental Technological Research, Polish Academy of Sciences, Poland
[4]Department of Engineering, Faculty of Science, Technology and Medicine, University of Luxembourg

***Abstract*—We present a distributed framework for predicting whether a planned reconfiguration step of a modular robot will mechanically overload the structure, causing it to break or lose stability under its own weight. The algorithm is executed by the modular robot itself and based on a distributed iterative solution of mechanical equilibrium equations derived from a simplified model of the robot. The model treats inter-modular connections as beams and assumes no-sliding contact between the modules and the ground. We also provide a procedure for simplified instability detection. The algorithm is verified in the Programmable Matter simulator *VisibleSim*, and in real-life experiments on the modular robotic system *Blinky Blocks*.**

***Keywords:* Self-reconfiguration, Modular robots, Distributed algorithms, Mechanical constraints, Programmable Matter.**

I. Introduction

Materials able to autonomously assume any shape are the dream of engineers. Currently, the most advanced artificial systems possessing elements of this functionality are modular self-reconfigurable robots—machines composed of robotic units (*modules*) which can bond together, move over one another and communicate, as well as store and process information [1]. They may be compared to swarms of fire ants forming bio-mechanical structures from their own bodies [2]. Cooperation of millions of tiny, densely-packed modules is expected to produce the desired emergent shape change of the entire ensemble. A system of this kind would be a realization of the futuristic concept of *Programmable Matter* [3].

The operation of densely-packed self-reconfigurable robots is based on the movement of modules from one part of the robot to another. This poses not only the challenging hardware problem of designing miniaturized modules able to operate in large 3D ensembles, but also the software problem of controlling this motion. If the robot is autonomous, its operation must be collectively planned and controlled by the modules, taking into account *geometric* and *mechanical* constraints on reconfiguration at each stage of motion. Although offline approaches could be used to check these constraints efficiently for a number of selected configurations, the less efficient online approaches must be used in general to handle any shape or any possible interaction with obstacles.

Geometric constraints result from the fact that modules need sufficient space to make a planned move but also must constantly remain in physical contact with other modules [4]. Mechanical constraints, in turn, result from the requirements of integrity and stability of the entire robot—the structure cannot break at inter-modular junctions or lose balance during reconfiguration. The mechanical constraints can be neglected in some special cases, like reconfiguration under no external loading (weightlessness) or 2D reconfiguration on flat ground with perpendicular gravity as the only loading. Otherwise, they usually need to be considered.

Currently, almost all algorithms for planning and controlling self-reconfiguration of densely-packed systems take only geometric constraints into account, like in [5], [6], [7], [8], [9]. An up-to-date survey of self-reconfiguration algorithms for modular robots can be found in [10]. By contrast, works on the mechanical behavior of densely-packed modular structures are few, present mostly centralized procedures, and rarely discuss reconfiguration. Examples can be found in [11], focused on optimizing the compliance of modular tools, and in [12], aimed at predicting the mechanical behavior of structures produced by additive manufacturing. As a separate research direction, special types of modular structures were investigated in [13], [14] and [15] to check the possibility of using modular robots as collective actuators. Autonomous reconfiguration planning that takes into account both geometric and mechanical constraints remains a challenging open problem, some aspects of which are investigated in this work.

In the present paper, we develop the approach introduced in [16] much further into a more realistic framework. Arbitrary 3D structures are investigated, for which a linear-elastic FE model is again adopted, with the addition of unilateral contact conditions which represent interaction with the surroundings. Two failure modes are considered: overloading of inter-modular connections and loss of balance, both checked in a distributed manner. Two methods of checking the loss of balance are proposed: (1) a simplified one, valid for structures standing on a flat surface, and (2) a model-based one, which is more general but requires solution of the mechanical balance equations with contact conditions. Verification is performed in a dedicated simulator *VisibleSim* [17], as well as experimentally on the real robotic modules *Blinky Blocks* [18]. The computational cost and several possible extensions of the applied weighted Jacobi iterative solver are also discussed.

II. Distributed prediction of the mechanical state of a robot

A. Basic characteristics of a modular robot

For the purposes of algorithmic mechanical analysis, a modular robot will be represented only by its connection topology and inter-modular connection strengths. We will focus on structures built of cubic *Blinky Blocks* [18], which are arranged on the Cartesian grid. However, in principle, the proposed approach can handle other connection topologies.

From the information-theoretic point of view, a modular robot has a distributed and asynchronous computing architecture, with mesh connection topology corresponding to the communication network of modules [19]. In *Blinky Blocks*, information can only be directly transferred between adjacent modules, so the communication network has the same connection topology as the modular robot itself. Such neighbor-to-neighbor-only communication imposes strict limits on the information-passing abilities of the system and requires special algorithms for effective prediction of mechanical failures.

B. Problem to be solved

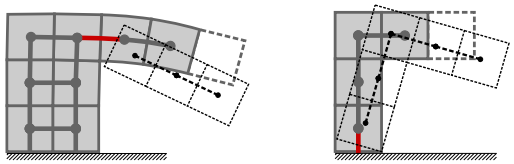
Reconfiguration of a modular robot can be broken down into simple steps. A single step consists of (a) releasing some inter-modular connections, (b) shifting modules which have been freed into nearby positions and, finally, (c) creating new connections at the new locations. As an alternative way of restructuring one may consider attaching new modules to a structure and discarding some old ones. Each reconfiguration step can potentially cause failure of the structure, which is in general irreversible. To avoid it, a prior mechanical analysis can be performed to ascertain that the planned new configuration is mechanically safe.

We restrict further presentation only to the case of attaching additional modules to an existing structure. Nevertheless, the whole idea can also be applied to a complete reconfiguration step. In such a case, the mechanical analysis would need to be performed for each of the above stages (a), (b) and (c) of the planned reconfiguration step. The step would be considered permissible if the structure was predicted to be safe after each of the three stages. Since this kind of reconfiguration cannot be easily performed on *Blinky Blocks*, we do not further develop this topic in the present paper.

Algorithmic prediction of two failure modes, shown in Fig. 1, will be analyzed, expanding on the ideas proposed in [16]. The first one is when addition of new blocks increases stresses at some inter-modular junctions beyond the holding capacity of connectors, as a result of which the structure breaks, Fig. 1a. The second one is when the structure loses balance when the new modules are added, Fig. 1b.

C. Overview of model-based failure prediction

We propose a distributed procedure which can predict both types of failure simultaneously. It approximately solves a special mechanical problem for the robot with new modules attached and can be customized to handle different module designs. The procedure has several distinctive features, described in detail in the succeeding sections:



(a) (b)

Fig. 1. Two types of failure after an additional module (gray dashed line) is attached: (a) breakage of a connection; (b) loss of overall stability. The red lines designate failing connections.

- The modular robot is represented by a Finite Element (FE) model; see Sec. II-D.
- Two types of connections are assumed: the inter-modular connection, modeled as a linear-elastic beam, and the connection between a module and an external support (e.g., the ground), modeled as a linear-elastic beam with unilateral no-sliding contact conditions at the support end.
- The mechanical state of a planned configuration with new modules attached (see Sec. II-E) is obtained by solving a non-linear problem (non-linearities result from the contact conditions; see Sec. II-F).
- The problem is solved in a distributed fashion using the weighted Jacobi iterative scheme (see Sec. II-G).
- After the iterations converge, two failure criteria are checked: for the loss of balance (see Sec. III-C) and overloading of connections (see Sec. III-D).

D. Standard 3D frame model

In the adopted FE model, each module p is represented by a node with 6 degrees of freedom $\mathbf{u}_p$ (3 displacements, u_x, u_y, u_z, and 3 rotations, τ_x, τ_y, τ_z) and each pair of connected modules is represented by a beam joining their nodes. A module in contact with the ground is represented by a special beam between the module's node and a "ground" node g, with $\mathbf{u}_g = \mathbf{0}$, as it is explained in Sec. II-F. In Fig. 1, the beams are presented as lines joining the centers of adjacent modules.

In a coordinate system CS whose z axis points upwards, the stiffness matrices for a beam joining module p and module q lying below it read $\mathbf{K}^{11}_{pq} = \mathbf{K}^{11}$ and $\mathbf{K}^{12}_{pq} = \mathbf{K}^{12}$, where:

$$\mathbf{K}^{11} = \frac{E}{L^3}\begin{pmatrix} 12I_x & 0 & 0 & 0 & -6I_xL & 0 \\ 0 & 12I_y & 0 & 6I_yL & 0 & 0 \\ 0 & 0 & AL^2 & 0 & 0 & 0 \\ 0 & 6I_yL & 0 & 4I_yL^2 & 0 & 0 \\ -6I_xL & 0 & 0 & 0 & 4I_xL^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & J_vL^2 \end{pmatrix}, \quad (1)$$

$$\mathbf{K}^{12} = \frac{E}{L^3}\begin{pmatrix} -12I_x & 0 & 0 & 0 & -6I_xL & 0 \\ 0 & -12I_y & 0 & 6I_yL & 0 & 0 \\ 0 & 0 & -AL^2 & 0 & 0 & 0 \\ 0 & -6I_yL & 0 & 2I_yL^2 & 0 & 0 \\ 6I_xL & 0 & 0 & 0 & 2I_xL^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -J_vL^2 \end{pmatrix}, \quad (2)$$

while E, L, A, I_x, I_y and J_v are the elastic modulus, length, cross-sectional area, area moments of inertia in the x and y directions, and scaled torsion constant in the z direction of the

symbol	value	description
E	100 MPa	elastic modulus
$L,\quad A=L^2$	40 mm, 40×40 mm^2	length & cross-sectional area
I_x, I_y	$L^4/12$ mm^4	x and y moments of inertia
$J_\nu=\frac{J}{2(1+\nu)}$	$2.25\cdot L^4/41.6$ mm^4	scaled z torsion constant (J: z torsion constant, ν: Poisson's ratio)
M	0.061 06 kg	mass of a block
$F_{\mathrm{V}}^{\max}$	11.98 N	strength of a vertical connection
$F_{\mathrm{L}}^{\max}$	14.97 N	strength of a lateral connection

TABLE I
BLINKY BLOCKS' GEOMETRIC AND MATERIAL PARAMETERS

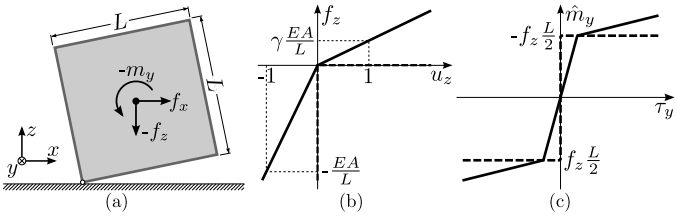


Fig. 2. Regularized contact conditions: (a) a module in contact with the ground, (b) contact-separation condition for the f_z component, and (c) tilting (stable/unstable) condition for the m_y component. The dashed lines indicate the respective "exact" (non-regularized) relationships for the case of a rigid module in contact with rigid ground.

beam, respectively (see Tab. I). If the two neighbors p and q are aligned with the z axis of another coordinate system CS', then $\mathbf{K}_{pq}^{11}$ and $\mathbf{K}_{pq}^{12}$ take the form:

$$\mathbf{K}_{pq}^{11}=\hat{\mathbf{R}}_{pq}\mathbf{K}^{11}\hat{\mathbf{R}}_{pq}^{\mathrm{T}}\quad,\quad\mathbf{K}_{pq}^{12}=\hat{\mathbf{R}}_{pq}\mathbf{K}^{12}\hat{\mathbf{R}}_{pq}^{\mathrm{T}},\tag{3}$$

where

$$\hat{\mathbf{R}}_{pq}=\begin{pmatrix}\mathbf{R}_{pq}&\mathbf{0}\\\mathbf{0}&\mathbf{R}_{pq}\end{pmatrix},\tag{4}$$

$\mathbf{R}_{pq}$ is the 3×3 rotation matrix from CS' to CS, $\square^{\mathrm{T}}$ denotes a transpose, and block matrix notation is used.

The ensemble is in static equilibrium if, for every module p, the sums of reaction forces and torques between p and all its neighbors q are equal to the gravitational force and null external torque acting on p, respectively, given by the vector $\mathbf{F}_p^{\mathrm{ext}}=[0,0,\text{-}9.81\cdot M,0,0,0]^{\mathrm{T}}$. The equilibrium equations are:

$$\forall_p\left[\sum_q\mathbf{K}_{pq}^{11}\mathbf{u}_p+\mathbf{K}_{pq}^{12}\mathbf{u}_q\right]=\mathbf{F}_p^{\mathrm{ext}},\tag{5}$$

with unknown vectors $\mathbf{u}_p$ and $\mathbf{u}_q$. After assembling the global vectors $\mathbf{u}$, $\mathbf{F}^{\mathrm{ext}}$ and stiffness matrix $\mathbf{K}$, incorporating all degrees of freedom of the structure, Eq. (5) takes the form $\mathbf{K}\mathbf{u}=\mathbf{F}^{\mathrm{ext}}$. It still needs to be extended to account for the modules planned to be added (Sec. II-E) and contact conditions (Sec. II-F). The respective quantities for the extended system, called the perturbed state, will be denoted by symbols with bars.

E. Adding virtual modules

To predict the state of the system one reconfiguration step ahead, the algorithm must take into account *virtual modules*—the modules which are planned to be attached. This is done in a simple way: a system analogous to Eq. (5) is built assuming that virtual modules are present. During computation, virtual modules are emulated by their existing neighbors, which store and process variables and messages related to virtual modules.

F. Contact with external supports

We use a simplified contact model of a cubic module with an external support, in which the support must be flat and coplanar with one of the module's facets. Also, we only analyze initially existing contact interfaces and assume that no new ones appear under load. We distinguish two conditions: (i) a unilateral contact-separation condition (coaxial mode) combined with no-sliding/no-twisting (shear and torsional modes), (ii) a tilting condition (bending mode). We say that a module is in contact if the axial force in the beam representing the contact is compressive. Otherwise, the module is in separation. When in contact, the module can only tilt if the bending torque in the beam exceeds a limit torque which is proportional to the compressive force (see the explanation below).

Without loss of generality, let us consider a module supported from below. The module is loaded with the forces and torques $\mathbf{F}=[f_x,f_y,f_z,m_x,m_y,m_z]^{\mathrm{T}}$, which correspond to the kinematic variables $\mathbf{u}=[u_x,u_y,u_z,\tau_x,\tau_y,\tau_z]^{\mathrm{T}}$. The contact condition (see Fig. 2b) takes the form of the Signorini problem:

$$f_z\le0\qquad\&\qquad u_z\ge0\qquad\&\qquad f_zu_z=0.\tag{6}$$

Additionally, we require that when the contact is active ($f_z<0$), there is no tangential slip ($u_x=0$ & $u_y=0$) or twisting ($\tau_z=0$), and the module can only tilt if at least one of the bending torques exceeds a limit torque. The tilting conditions (see Fig. 2c) can be conveniently written in the following form:

$$\begin{aligned}&\Phi_x\le0\quad\&\quad\tau_x\cdot\mathrm{sign}(\hat{m}_x)\ge0\quad\&\quad\Phi_x\cdot\tau_x=0,\\&\text{where}\quad\hat{m}_x=m_x-f_y\cdot L/2,\quad\Phi_x=|\hat{m}_x|+f_z\cdot L/2;\end{aligned}\tag{7}$$

$$\begin{aligned}&\Phi_y\le0\quad\&\quad\tau_y\cdot\mathrm{sign}(\hat{m}_y)\ge0\quad\&\quad\Phi_y\cdot\tau_y=0,\\&\text{where}\quad\hat{m}_y=m_y+f_x\cdot L/2,\quad\Phi_y=|\hat{m}_y|+f_z\cdot L/2;\end{aligned}\tag{8}$$

where L is the module's size. The tilting (bending) condition expresses the fact that the torques $|\hat{m}_x|$ or $|\hat{m}_y|$ cannot exceed the torque $\text{-}f_z\cdot L/2$ produced by the compressive force $\text{-}f_z$ about any of the facet's edges; see also Fig. 2a.

A supported module is modeled as a beam with the same elastic constants as two connected modules. This gives the force-displacement relationship for the case of stable contact (when the module adheres to the support with an entire facet). This relationship is then used by a special predictor-corrector scheme, described below, which provides regularization to the contact conditions given by Eqs. (6–8).

In the regularized contact scheme for module p, a trial state is computed first, assuming a linear-beam-type connection with each support q (here, the contact direction is z):

$$\mathbf{F}_{pq}^{\mathrm{tr}}=[f_x^{\mathrm{tr}},f_y^{\mathrm{tr}},f_z^{\mathrm{tr}},m_x^{\mathrm{tr}},m_y^{\mathrm{tr}},m_z^{\mathrm{tr}}]^{\mathrm{T}}=\mathbf{K}_{pq}^{11}\bar{\mathbf{u}}_p,\tag{9}$$

where the term $\mathbf{K}_{pq}^{12}\bar{\mathbf{u}}_q$ is absent because the support is assumed to be immobile, $\bar{\mathbf{u}}_q=\mathbf{0}$. Then, a corrected vector $\mathbf{F}_{pq}$ is determined, taking into account two conditions:

(i) the unilateral (normal) contact-separation condition combined with no-sliding and no-twisting conditions:

$$\begin{cases} \mathbf{F}_{pq} := \gamma \cdot \mathbf{F}^{\mathrm{tr}}_{pq} & \text{for } f^{\mathrm{tr}}_z \geq 0 \quad \text{(separation)} \\ (f_x, f_y, f_z, m_z) = (f^{\mathrm{tr}}_x, f^{\mathrm{tr}}_y, f^{\mathrm{tr}}_z, m^{\mathrm{tr}}_z) & \text{otherwise} \quad \text{(contact)} \end{cases} \tag{10}$$

(ii) the tilting (bending) condition, used only when $f^{\mathrm{tr}}_z < 0$, i.e. when the module is in contact:

$$m_x := \begin{cases} m^{\mathrm{tr}}_x & \text{for } \Phi^{\mathrm{tr}}_x < 0 \text{ (stable)} \\ \gamma \cdot m^{\mathrm{tr}}_x + \bar{\gamma} \cdot \frac{L}{2} \left[f^{\mathrm{tr}}_y - \mathrm{sign}\left(\hat{m}^{\mathrm{tr}}_x\right) f^{\mathrm{tr}}_z \right] & \text{otherwise (tilting)} \end{cases} \tag{11}$$

$$m_y := \begin{cases} m^{\mathrm{tr}}_y & \text{for } \Phi^{\mathrm{tr}}_y < 0 \text{ (stable)} \\ \gamma \cdot m^{\mathrm{tr}}_y - \bar{\gamma} \cdot \frac{L}{2} \left[f^{\mathrm{tr}}_x + \mathrm{sign}\left(\hat{m}^{\mathrm{tr}}_y\right) f^{\mathrm{tr}}_z \right] & \text{otherwise (tilting)} \end{cases} \tag{12}$$

where $\hat{m}^{\mathrm{tr}}_x$, $\hat{m}^{\mathrm{tr}}_y$, Φ^{tr}_x and Φ^{tr}_y are computed as in Eqs. (7) and (8), but using the components of $\mathbf{F}^{\mathrm{tr}}_{pq}$; $\gamma = 10^{-4}$ and $\bar{\gamma} = 1 - \gamma$.

The corrected contact tangent matrix $\bar{\mathbf{K}}^{11}_{pq}$ is obtained as a derivative of $\mathbf{F}_{pq}$ with respect to $\bar{\mathbf{u}}_p$. Again, the matrix $\bar{\mathbf{K}}^{12}_{pq}$ is disregarded because supports are assumed to be immobile.

Remark: Ill-posedness of the problem is avoided by introducing a very weak spring, characterized by γ, that prevents free rigid-body motion of the structure. The drawback of this approach is poor conditioning of the resulting system of equations when the robot is unstable, which can deteriorate the convergence rate of the iterative solver; see Sec. II-H. However, as it is shown in Sec. III-C, the knowledge of which supports are active suffices for assessing stability and those are usually identified long before convergence is achieved.

G. Weighted Jacobi solution scheme

The global system of equations for the perturbed system, in which virtual modules are included in the model (Sec. II-E) and contact conditions are accounted for by the predictor-corrector scheme (Sec. II-F), reads:

$$\bar{\mathbf{K}}\bar{\mathbf{u}} = \bar{\mathbf{F}}^{\mathrm{ext}}. \tag{13}$$

Eq. (13) is solved iteratively using the weighted Jacobi scheme [16]. A single iteration $i \to i+1$ for module p reads:

$$\bar{\mathbf{u}}^{i+1}_p = \beta \bar{\mathbf{D}}_p{}^{-1} \left(\bar{\mathbf{F}}^{\mathrm{ext}}_p - \bar{\mathbf{R}}_p \bar{\mathbf{u}}^i_p - \sum_q \bar{\mathbf{K}}^{12}_{pq} \bar{\mathbf{u}}^i_q \right) + (1-\beta)\bar{\mathbf{u}}^i_p, \tag{14}$$

where $\bar{\mathbf{D}}_p = \mathrm{diag}\left(\sum_q \bar{\mathbf{K}}^{11}_{pq}\right)$ and $\bar{\mathbf{R}}_p = \left(\sum_q \bar{\mathbf{K}}^{11}_{pq}\right) - \bar{\mathbf{D}}_p$ are the diagonal and the remainder parts of the respective stiffness sub-matrices, while $\beta = 2/3$. Initially, we set $\bar{\mathbf{u}}^0 = \mathbf{0}$ (alternatively, the solution for the non-perturbed state, if available, could be used instead of $\mathbf{0}$, which would reduce the necessary number of iterations). Note that in the iteration $i+1$ only the values of $\bar{\mathbf{u}}$ from the iteration i are used, and only those from p and its direct neighbors q. Thus only local communication is involved and the memory complexity is constant.

In the present implementation, the weighted Jacobi procedure is initiated by the centroid module which sends an *Init* message down the spanning tree, broadcasting the number of iterations to be done (see Sec. III-B for the centroid module and the spanning tree). Having received the *Init* message, each *Blinky Block* B_p sends its initial vector $\bar{\mathbf{u}}^0_p = \mathbf{0}$ to all its neighbors and initializes its iteration counter, $\mathrm{iter}_p = 0$. In a given iteration $\mathrm{iter}_p = i$, B_p can receive from any of its neighbors B_q a message containing $\bar{\mathbf{u}}^i_q$ (displacements of B_q calculated in iteration i), which is then stored in B_p's buffer. When B_p has received $\bar{\mathbf{u}}^i_q$ from all its neighbors, it computes $\bar{\mathbf{u}}^{i+1}_p$ (see Eq. 14), increments its counter, $\mathrm{iter}_p \leftarrow \mathrm{iter}_p + 1$, and sends $\bar{\mathbf{u}}^{i+1}_p$ to all its neighbors. The process continues until the prescribed number of iterations is reached.

Remark. The weighted Jacobi procedure behaves like the Alpha local synchronizer [20].

	$\mathbb{C}_{\mathrm{C}}$	$\mathbb{C}_{\mathrm{T}}$, $\mathbb{C}_{\mathrm{M}}$, $\mathbb{C}_{\mathrm{SST}}$, $\mathbb{C}_{\mathrm{MST}}$	$\mathbb{C}_{\mathrm{WJ}}$
Execution time	$O(d)$	$O(\tilde{d})$	$O(n^2)$
No. of CPU operat./module	$O(1)$	$O(1)$	$O(n^2)$
Total no. of messages sent	$O(n)$	$O(n)$	$O(n^3)$
Memory usage per module	$O(1)$	$O(1)$	$O(1)$

TABLE II
COMPLEXITY ASSESSMENTS FOR SUBROUTINES OF THE ALGORITHM. $\mathbb{C}_{\mathrm{C}}$ REFERS TO CENTROID SELECTION, $\mathbb{C}_{\mathrm{T}}$—TO TREE CONSTRUCTION, $\mathbb{C}_{\mathrm{M}}$—TO FINDING THE CENTER OF MASS, $\mathbb{C}_{\mathrm{SST}}$—TO THE SIMPLIFIED STABILITY CHECK, $\mathbb{C}_{\mathrm{MST}}$—TO THE MODEL-BASED STABILITY CHECK, AND $\mathbb{C}_{\mathrm{WJ}}$—TO THE WEIGHTED JACOBI PROCEDURE.

H. Convergence properties and possible improvements

The Weighted Jacobi scheme converges if the spectral radius of the iteration matrix $\mathbf{C}_\beta = \mathbf{I} - \beta \bar{\mathbf{D}}^{-1} \bar{\mathbf{K}}$ is less than 1:

$$\rho(\mathbf{C}_\beta) = \max(|\lambda_1|, \ldots, |\lambda_N|) < 1, \tag{15}$$

where λ_i are the eigenvalues of $\mathbf{C}_\beta$. Although in the cases analyzed here convergence is achieved, the number of iterations is very high, which is a well-known drawback of the method (the convergence rate tends to 1 when the system grows [21]).

For the one-dimensional spring-in-series system of size n, we have analytically assessed the number of iterations necessary to attain an arbitrary relative error to be $O(n^2)$. This is also confirmed numerically in Sec. IV-B for more complex structures. The assessment shows the low efficiency of the scheme and underlies the complexities provided in Table II.

The framework presented in this work is not restricted to the weighted Jacobi scheme though. Its efficiency can potentially be significantly improved by adapting another method to solving the considered contact problem in a distributed way. We will briefly outline the three most promising directions.

Direction 1: The Krylov subspace methods [22] guarantee that the maximal number of iterations is at most equal to the number of degrees of freedom of the system, if the problem to be solved is linear. This can be further improved by appropriate preconditioning (see our preliminary study [23]). However, the need for global data aggregation and the non-linearities introduced by the contact problem require special treatment, which can deteriorate the time and memory efficiency.

Direction 2: Multigrid techniques [24] can more easily capture long-wave modes of the solution, which should improve the convergence rate. A special version must be devised, however, taking into account the contact conditions (like the one recently proposed [25]) and the specific computing architecture of the modular robot.

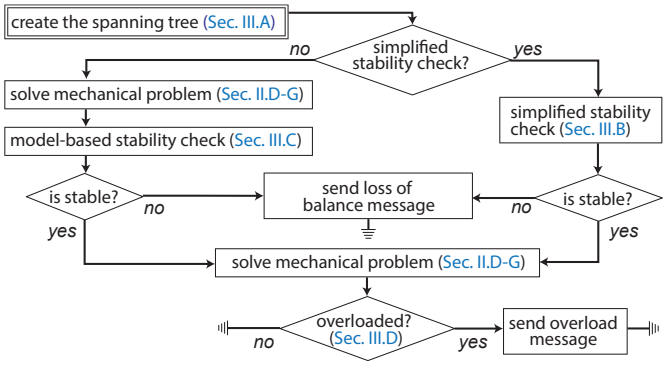


Fig. 3. Flowchart of the algorithm.

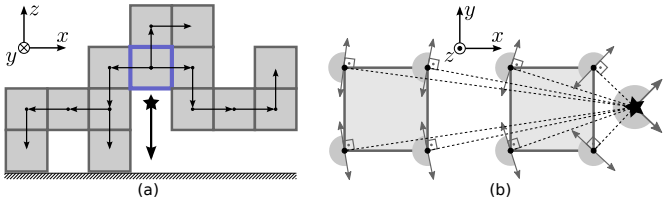


Fig. 4. (a) A modular robot (unstable) with a selected centroid (thick blue line), the center of mass (star) and a spanning tree (arrows). (b) Detection of whether a given point (star) is inside the convex hull of the support points. The condition is fulfilled if and only if all the straight angles (shaded) sum up to the full angle (they do not in this example).

Direction 3: The number of degrees of freedom of the system can be reduced by applying multi-scale methods or model order reduction techniques [26], [27]. However, it may be hard to find a suitable reduced space online efficiently.

III. Mechanical Stability and Overload Check

Below we describe computational methods using which a robot can autonomously predict the two types of failure shown in Fig. 1, one reconfiguration step ahead. The methods utilize a spanning tree, which is discussed first in Sec. III-A. Sec. III-B describes a simple method of checking stability—without iterations, but restricted to robots standing on flat ground. Sec. III-C discusses stability verification in the general case, using the iterative scheme of Sec. II. Finally, in Sec. III-D, conditions for inter-modular connection breakage are presented, utilizing the results of the iterative scheme. The flowchart of the procedure is shown in Fig. 3.

Assessments of complexities of the subroutines of the algorithm are presented in Table II, where n is the number of modules, d is the radius of the connection graph of the robot, and $\tilde{d}$ is the depth of the spanning tree (usually, $\tilde{d}$ and d are of the same order; see Sec. III-A for more details).

A. Spanning tree

A spanning tree allows efficient communication inside the robot. Its construction begins with a choice of the *centroid* module, serving as the root, which is selected near the robot's topological center; see e.g. Fig. 4a. This can be done automatically [28], but we chose the centroid manually in all examples.

The tree is extended to all modules, starting at the centroid which sends a *Tree* message to its neighbours. When a module receives the *Tree* message for the first time, it becomes a next-level node and sends the *Tree* message further. This usually leads to the construction of BFS-like trees of quasi-optimal depth, without the need for synchronization. However, in rare cases the resulting tree may be far from optimal, with a Hamiltonian path over the structure as the worst-case scenario. The algorithms relying on the tree are then adversely affected.

B. Loss of balance in a simplified case

It is assumed that the robot is rigid and stands on flat ground under vertical gravity (Fig. 4a). The modules know their own masses and positions in a common Cartesian coordinate system, with $z = 0$ being the ground level. They also simulate the behavior of their virtual neighbors (Sec. II-E).

The stability check reduces to verifying that the center of mass of the robot lies over the convex hull of the points of support. If it does, the robot is stable, otherwise, it is not. The algorithm proceeds as follows.

(a) The center of mass of the robot is computed. Starting at the leaves of the spanning tree, each node sums up the masses of all its subbranches and its own, m_i, and likewise the weighted centers of mass of all its subbranches and its own, $m_i\mathbf{X}_i$. The two sums are propagated to the parent node and the process continues. At the centroid, the center of mass of the robot is retrieved as $[X,Y,Z] = (\Sigma m_i \mathbf{X}_i)/(\Sigma m_i)$.

(b) X and Y are broadcast over the tree.

(c) For any supporting module i, its *safe angle range* $[\alpha_i, \beta_i]$ is determined as the sum of safe angle ranges of its corners. The 180° safe angle range of a corner $\mathbf{p}_j = [X_j, Y_j, 0]$ is swept by the planar vector $[X_j - X, Y_j - Y]$ when turning 90° left or right. It covers those directions in which the structure cannot tilt; see Fig. 4b. The safe angle range of any corner $\mathbf{p}_j = [X_j, Y_j, Z_j \neq 0]$ is assumed to be empty.

(d) The safe angle ranges are summed up over the tree, just like masses were in step (a). The summation always gives a single interval, because all considered ranges are either empty or not less than the straight angle.

(e) The structure is stable if and only if the aggregated angle range at the centroid equals the full angle.

C. Loss of balance in the model-based approach

In the general case with arbitrarily placed supports, there is no simple method to predict stability. Sometimes, if a model of the robot is simple, like under the rigid-body or elasto-static assumptions used here, there may even be no unique answer. Since more accurate modeling goes beyond the scope of the present paper, we will show how to utilize the proposed elasto-static model with contact and the iterative solution scheme to check stability in more general cases. The method, however, does not assure finding the correct solution in difficult cases (e.g., when the solution is non-unique by definition).

The method is based on the observation that, when the iterative solution scheme has converged, the local state of the contact conditions is fully determined. It is only necessary to check whether active supports prevent rigid-body rotation of the structure (at least three noncollinear support points must be active), which is achieved by the following procedure:

(a) Solve the mechanical problem (see Sec. II) and initiate the stability check (the spanning tree is used again).
(b) For every module, determine the set of its active corner points by checking the contact conditions (Sec. II-F):
  - No contact → return the empty set.
  - Tilting in two directions → return a single corner—the common point of the two edges of rotation.
  - Tilting in one direction → return two corners—the end points of the edge of rotation.
  - Contact & no tilting → return a special 'stable' state.
(c) Aggregate information starting at the leaves and moving up the spanning tree towards the centroid:
  - At each module, sum the sets of active points of its subbranches and its own, obtaining set S. By convention, adding any set to 'stable' gives 'stable'.
  - If the points in S are noncollinear then set $S =$ 'stable'.
  - If multiple points in S are collinear then leave only two.
  - Pass S up the spanning tree.
(d) The stability check ends at the centroid, with the result being either 'stable' or not.

Excluding the expensive phase of determining active supports (weighted Jacobi iterations), the complexity of the approach is the same as that of the simplified case; see Table II. The memory complexity per module is constant because each returned set of points has at most two elements.

D. Overloading of inter-modular connections

Connection overloading is checked when the iterative scheme has sufficiently converged and after checking that the structure is stable. The forces and torques which act between module p and its neighbor q are predicted as follows:

$$[f_x, f_y, f_z, m_x, m_y, m_z]^{\mathrm{T}} = \frac{1}{2}\hat{\mathbf{R}}_{pq}^{\mathrm{T}}(\mathbf{F}_{pq} - \mathbf{F}_{qp}) = \\ = \frac{1}{2}\hat{\mathbf{R}}_{pq}^{\mathrm{T}}(\bar{\mathbf{K}}_{pq}^{11}\bar{\mathbf{u}}_p + \bar{\mathbf{K}}_{pq}^{12}\bar{\mathbf{u}}_q - \bar{\mathbf{K}}_{qp}^{11}\bar{\mathbf{u}}_q - \bar{\mathbf{K}}_{qp}^{12}\bar{\mathbf{u}}_p), \quad (16)$$

where $\hat{\mathbf{R}}_{pq}^{\mathrm{T}}$ rotates the resulting vector into a coordinate system in which axial forces are aligned with the z axis.

To avoid connection breakage, the tensile force f_z and torques m_x and m_y, computed in Eq. (16) in the middle of the connection, must not overpower the magnetic forces $F^{\max}$ binding the modules. (Shear and torsion are omitted, because *Blinky Blocks*' connectors are assumed to be resistant to those modes of breakage.) The vertical and lateral connections of *Blinky Blocks* differ, so that $F^{\max}$ can take two values; see Table I. The safety condition for both tension and bending is:

$$F^{\max} > 2 \cdot \max(|m_x|, |m_y|)/L + f_z. \quad (17)$$

The check is performed for all connections and the results are aggregated by the centroid over the spanning tree.

IV. Implementation, simulations and experiments

A. Implementation details

The procedures have been implemented and tested in the integrated environment developed at FEMTO-ST[1]. It consists of the virtual test bed *VisibleSim* [17] and the reconfigurable modular robot *Blinky Blocks* [18], so the same implementation could be executed on both platforms. The software was adjusted to be compatible with the real *Blinky Blocks* Version 1 hardware: reduced floating-point precision of 4 Bytes was used and the maximum message size was set to 17 Bytes (messages containing 6×4 Byte-long vectors were split in half).

The program's flowchart is shown in Fig. 3, and the consecutive steps of the algorithm are discussed in the previous sections. The choice between the simplified (Sec. III-B) and the full (Sec. III-C) stability check to be performed is preset by the user. In the case of the *Blinky Blocks* hardware, a preliminary step is additionally performed, in which the same main program is loaded into each *Blinky Block*, and a common coordinate system for a given configuration of *Blinky Blocks* is propagated, starting from a special block with preset coordinates.

The material parameters of the *Blinky Block* model described in Sec. II-D are provided in Table I. All have been assessed experimentally, except Young's modulus which was chosen arbitrarily (its exact value is not essential for assessing overload and stability). The dimensions and mass have been measured directly. Connection strengths have been obtained in a simplified manner, by finding the maximum number of *Blinky Blocks* that their magnets could hold hanging in a vertical alignment. The top/bottom and lateral connection strengths differ because the former is produced by a Lego-like system reinforced with a central magnet, and the latter by 4 magnets placed in the corners of each face; see also Fig. 5.

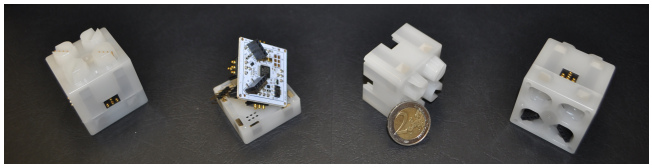


Fig. 5. *Blinky Blocks*: functional ones (left and right), and two pieces of a dismantled one with a top view of the motherboard (in the middle).

B. Simulations and experiments

Six different modular configurations and failure scenarios were investigated (see Fig. 6), both in *VisibleSim* and on *Blinky Blocks*[2]. The sizes of structures ranged from 8 modules in test #1(a) to 29 modules in test #6(c). Experiments on substantially larger structures would be difficult due to the high computational complexity of the algorithm combined with the limited processing and communication speed of the Version 1 of *Blinky Blocks*. In the physical experiments, reconfiguration of *Blinky Blocks* was done by manually attaching new modules to an existing structure. In all the presented cases, the model-based analysis was involved (addressing both overloading and loss of stability), which required execution of the weighted Jacobi iterative scheme. Additionally, in the loss-of-stability scenarios (Fig. 6 #1-#3), the results obtained with the simplified and the model-based stability checks were successfully cross-validated.

[1]Programmable Matter project at FEMTO-ST: https://projects.femto-st.fr/programmable-matter/.

[2]Videos of selected experiments: https://youtu.be/d3aE8GjbYd8

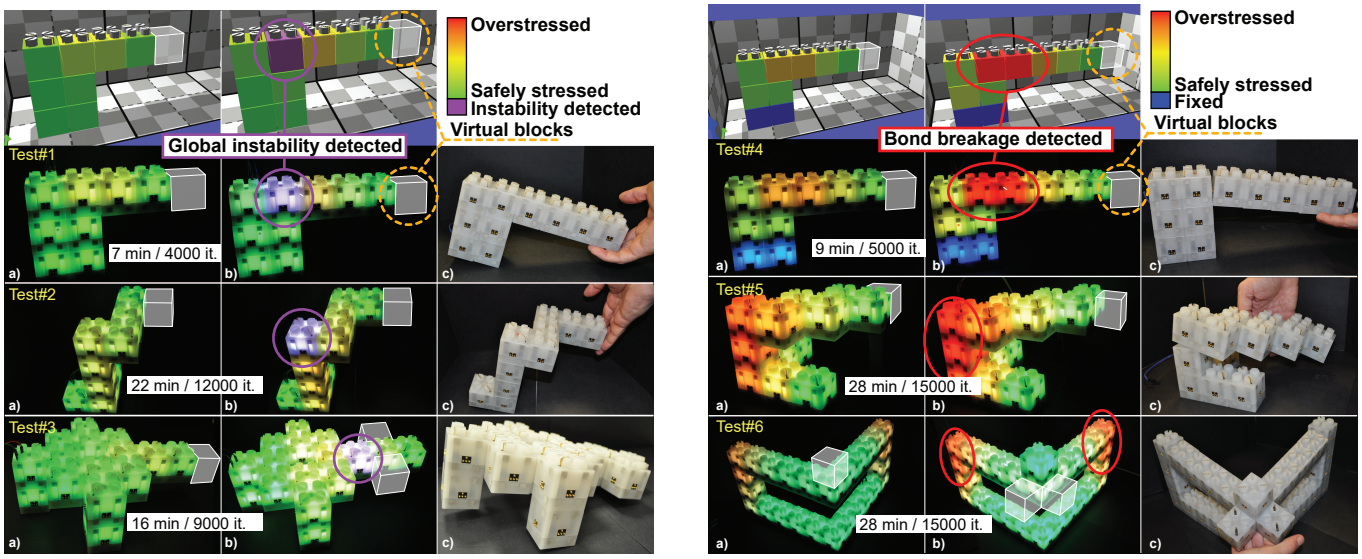


Fig. 6. Experiments in *VisibleSim* and on *Blinky Blocks*. Computation times and iteration numbers are shown in insets.

Because it is in general difficult to automatically assess the necessary number of weighted Jacobi iterations, this number was adjusted manually case by case. The criterion was to make the number of iterations possibly low while obtaining correct predictions at the same time. The problem of how the necessary number of iterations scales with the system size in a stable case is briefly discussed later. In an unstable case, this number is expected to be much higher. Following the conclusion in the final Remark of Sec. II-F, in unstable cases we stopped computations just after all contact states stabilized, but before numerical convergence was achieved.

Each of the six tests in Fig. 6 shows the results of execution of the program for two consecutive construction steps of a particular *Blinky Blocks* structure. In every figure, the first construction step (a→b) is designed to be mechanically safe, while the second one (b→c) to result in failure, which is then demonstrated in the third part of the figure (c). Additionally, in the top row, *VisibleSim* results are shown for the tests #1 and #4. The tests are split into loss-of-stability (left column) and overloading (right column) scenarios. From top to bottom, the scenarios are ordered by complexity, i.e., 2D cases in tests #1 & #4, 3D cases in tests #2 & #5, and 3D cases with more complex connection topologies in tests #3 & #6.

The results of calculations are displayed using colors: the color of a block corresponds to the highest tensile/bending stress level in any of its connections, as given by the right hand side of Eq. (17). Green to orange colors represent the safe stress range, while red indicates potentially overstressed connections. *Blinky Blocks* were programmed to blink in red when tensile stresses in some of their connections exceeded the critical level, while global imbalance of a structure was signaled by the centroid module blinking in purple. Blue *Blinky Blocks* are fixed—they are attached to the floor.

In all tests except #6, the predictions are confirmed by physical experiments. In test #6, breakage is correctly predicted but ill-localized. This can be observed in Fig. 6-#6(b) which

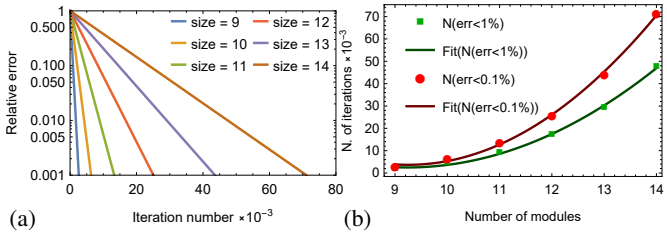


Fig. 7. Number of weighted Jacobi iterations necessary to attain a given accuracy for fixed-arms of different sizes. Size 10 refers to the example in Fig. 6-#4(a); other sizes have shorter/longer arms. (a) Convergence characteristics. (b) A quadratic polynomial fit for two accuracy levels.

indicates breakage of the pillars, while the actual breakage occurs as it is shown in Fig. 6-#6(c). There are two possible reasons for the observed discrepancy. The first one is that the assumed mechanical model of the modular robot is too simple. The second one is the omission of twisting torques from the adopted criterion of breakage. It was also very difficult to keep the structures #6(a) and #6(b) operational—an effect which was not expected. In both cases, weight-induced deformations caused separation of electrical connectors, despite the structures did not break. This necessitated using additional supports just to perform computations. We view test #6 as one of benchmark cases for future research on more accurate models and failure criteria.

CPU time and convergence properties. Computing $\bar{\mathbf{u}}_p^i$ and exchanging messages with neighbours takes a *Blinky Block* a nearly constant time $T \approx 110.5$ ms (9.05 runs per second). Because communication is local, T is also the global time of a single weighted Jacobi iteration, independent of the configuration. Since the time cost of the other steps of the algorithm is negligible (see Tab. II), the overall execution time can be assessed by multiplying the number of iterations by T.

The number of iterations needed to attain a given accuracy

greatly depends on the system's configuration and generally grows with the number of modules n. Assessment of the number of iterations is generally not straightforward, even without considering unilateral contact conditions; see also Sec. II-H. In Fig. 7 we demonstrate the expected trends for a given family of configurations. Fig. 7a shows linear convergence of the relative error $\|\bar{\mathbf{u}}^i - \bar{\mathbf{u}}^*\|/\|\bar{\mathbf{u}}^*\|$ as the number of iterations i grows, where $\bar{\mathbf{u}}^*$ is the numerically exact solution. Fig. 7b presents the necessary numbers of iterations from Fig. 7a for two example relative errors, displaying quadratic growth with n and confirming the assessments in Tab. II.

V. Conclusions and future research

We presented a distributed algorithm for checking if a modular robot will retain its mechanical integrity and stability after new modules are attached to it at prescribed positions. The algorithm can be used to assess the mechanical safety of a reconfiguration step planned by a self-reconfigurable robot. The procedure is designed to run on the modular robot itself, and we have verified its predictions through tests in the dedicated simulator *VisibleSim* and on the real modular system *Blinky Blocks*. To our knowledge, this is the first time three-dimensional modular-robotic structures compute their mechanical state in a fully distributed manner.

The algorithm can be improved towards: adopting faster iterative schemes, as discussed in Sec. II-H and tried in [23]; extending the application range to soft modular robots; checking the construction/reconfiguration several steps ahead; as well as addressing other module geometries and broader module-support contact scenarios. Future experimental validation will use the currently produced new version of Blinky Blocks with faster CPUs and communication, and possibly quasi-spherical catoms [29] having up to 12 neighbors per module and electrostatic connectors.

Acknowledgement. This work was partially supported by the EU Horizon 2020 Marie Sklodowska Curie Individual Fellowship *MOrPhEM* (grant No. 800150), by the NCN Project "Micromechanics of Programmable Matter" (grant No. 2011/03/D/ST8/04089), by the ANR (ANR-16-CE33-0022-02), the French Investissements d'Avenir program, ISITE-BFC project (ANR-15-IDEX-03), EIPHI Graduate School (contract ANR-17-EURE-0002), Mobilitech project and EU Horizon 2020 research and innovation programme (grant No. 811099 TWINNING Project DRIVEN for the Univ. of Luxembourg).

References

[1] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems," *IEEE Robotics Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[2] N. J. Mlot, C. A. Tovey, and D. L. Hu, "Fire ants self-assemble into waterproof rafts to survive floods," *Proceedings of the National Academy of Sciences*, vol. 108, no. 19, pp. 7669–7673, 2011.

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

[4] A. Nguyen, L. J. Guibas, and M. Yim, "Controlled module density helps reconfiguration planning," in *Proceedings of the 4th International Workshop on Algorithmic Foundations of Robotics*, pp. 23–36, 2000.

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

[6] K. Støy, "Using cellular automata and gradients to control self-reconfiguration," *Robotics and Autonomous Systems*, vol. 54, no. 2, pp. 135–141, 2006.

[7] B. Piranda and J. Bourgeois, "A distributed algorithm for reconfiguration of lattice-based modular self-reconfigurable robots," in *Proceedings of the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pp. 1–9, IEEE, 2016.

[8] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein, "A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots," in *15th IEEE International Symposium on Network Computing and Applications*, pp. 254–263, 2016.

[9] J. Lengiewicz and P. Hołobut, "Efficient collective shape shifting and locomotion of massively-modular robotic structures," *Autonomous Robots*, vol. 43, no. 1, pp. 97–122, 2019.

[10] P. Thalamy, B. Piranda, and J. Bourgeois, "A survey of autonomous self-reconfiguration methods for robot-based programmable matter," *Robotics and Autonomous Systems*, vol. 120, p. 103242, 2019.

[11] 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, pp. 103–121, 2011.

[12] J. Hiller and H. Lipson, "Dynamic simulation of soft multimaterial 3d-printed objects," *Soft robotics*, vol. 1, no. 1, pp. 88–101, 2014.

[13] J. Campbell and P. Pillai, "Collective actuation," *International Journal of Robotics Research*, vol. 27, no. 3–4, pp. 299–314, 2008.

[14] P. Hołobut, M. Kursa, and J. Lengiewicz, "Efficient modular-robotic structures to increase the force-to-weight ratio of scalable collective actuators," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3302–3307, 2015.

[15] J. Lengiewicz, M. Kursa, and P. Hołobut, "Modular-robotic structures for scalable collective actuation," *Robotica*, vol. 35, pp. 787–808, 2017.

[16] P. Hołobut and J. Lengiewicz, "Distributed computation of forces in modular-robotic ensembles as part of reconfiguration planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2103–2109, 2017.

[17] B. Piranda, "Visiblesim: Your simulator for programmable matter," in *Algorithmic Foundations of Programmable Matter, Dagstuhl Seminar 16271*, p. 12, 2016.

[18] 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*, CHI EA '11, (New York, NY, USA), pp. 1111–1116, ACM, 2011.

[19] A. Tanenbaum, *Structured Computer Organization*. Pearson Prentice Hall, 5th ed., 2006.

[20] M. Raynal, *Distributed algorithms for message-passing systems*, vol. 500. Springer, 2013.

[21] G. Strang, *Linear algebra and its applications*. Thomson, Brooks/Cole, 2006.

[22] Y. Saad, *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2nd ed., 2003.

[23] J. Lengiewicz, S. P. A. Bordas, and P. Hołobut, "Autonomous model-based assessment of mechanical failures of reconfigurable modular robots with a conjugate gradient solver," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 11696–11702, 2020.

[24] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*. SIAM, 2000.

[25] T. A. Wiesner, A. Popp, M. W. Gee, and W. A. Wall, "Algebraic multigrid methods for dual mortar finite element formulations in contact mechanics," *International Journal for Numerical Methods in Engineering*, vol. 114, no. 4, pp. 399–430, 2018.

[26] P. Kerfriden, P. Gosselet, S. Adhikari, and S. P. A. Bordas, "Bridging proper orthogonal decomposition methods and augmented newton–krylov algorithms: an adaptive model order reduction for highly nonlinear mechanical problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, pp. 850–866, 2011.

[27] L. Beex, P. Kerfriden, T. Rabczuk, and S. P. A. Bordas, "Quasi-continuum-based multiscale approaches for plate-like beam lattices experiencing in-plane and out-of-plane deformation," *Computer Methods in Applied Mechanics and Engineering*, vol. 279, pp. 348–378, 2014.

[28] A. Naz, B. Piranda, S. C. Goldstein, and J. Bourgeois, "Approximate-centroid election in large-scale distributed embedded systems," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 548–556, 2016.

[29] 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, 2018.