Autonomous model-based assessment of mechanical failures of reconfigurable modular robots with a Conjugate Gradient solver

Paweł Hołobut[1], Stéphane P.A. Bordas[2] and Jakub Lengiewicz[1,2]

***Abstract*— Large-scale 3D autonomous self-reconfigurable modular robots are made of numerous interconnected robotic modules that operate in a close packing. The modules are assumed to have their own CPU and memory, and are only able to communicate with their direct neighbors. As such, the robots embody a special computing architecture: a distributed memory and distributed CPU system with a local message-passing interface. The modules can move and rearrange themselves changing the robot's connection topology. This may potentially cause mechanical failures (e.g., overloading of some inter-modular connections), which are irreversible and need to be detected in advance. In the present contribution, we further develop the idea of performing model-based detection of mechanical failures, posed in the form of balance equations solved by the modular robot itself in a distributed manner. A special implementation of the Conjugate Gradient iterative solution method is proposed and shown to greatly reduce the required number of iterations compared with the weighted Jacobi method used previously. The algorithm is verified in a virtual test bed—the *VisibleSim* emulator of the modular robot. The assessments of time-, CPU-, communication- and memory complexities of the proposed scheme are provided.**

I. Introduction

Modular self-reconfigurable robots are mechatronic systems which can autonomously change their structure, thus potentially adapting to different environments and assuming different functionalities. They are composed of elementary robotic units—*modules*—which work together and form a compound system of variable connection topology [1]. Individual modules are usually relatively simple compared with typical robots, but possess several features which allow them to work collectively. In particular, they can attach to other modules and move over them, communicate with adjacent modules, and process information stored in their internal memory. In other words, they are small computers with limited resources, which also have some locomotion capabilities. A self-reconfigurable robot made of a huge number of microscopic modules could be considered an example of *Programmable Matter* [2].

One class of self-reconfigurable robots, which are of special interest to our investigations, are densely-packed spatial ensembles. The modules of such systems can be arranged arbitrarily in the volume they occupy (like grains of sand) or regularly, following strict attachment patterns (lattice-based systems). Reconfiguration and general shape-change of such robots is accomplished through relocation of part of their modules from one area of the robot to another, which often requires complex planning. A recent survey of self-reconfiguration planning algorithms for modular robots can be found in [3]. Individual approaches vary depending on the considered module geometry and assumptions about the lattice structure. Examples include reconfiguration of a porous structure arranged on a cubic lattice guided by attraction gradients [4], [5]; reconfiguration by hole propagation [6]; efficient distributed reconfiguration of planar square- [7] and hexagonal- [8] lattice-based systems; finally, reconfiguration by transporting modules through tunnels inside the structure, discussed in [9] and [10].

The above-mentioned reconfiguration methods are all purely geometric—the motion of modules satisfies only geometric constraints requiring that moving modules adhere to other modules, have sufficient space to move [11], and that the entire structure remain connected. For any of such methods to be workable, however, additional constraints of mechanical nature have to be considered. In a real physical setting, with a limited connection strength between modules and under the action of gravity and possibly other external loadings, reconfiguration may fail for at least two mechanical reasons [12]. The first one is loss of stability: when the center of mass of a modular robot shifts during reconfiguration in such a way that the structure is thrown out of balance and falls. The second one is breakage of inter-modular connections: when some of the connections between modules become overloaded during reconfiguration and split. Both of these modes of mechanical failure should be taken into account during reconfiguration planning, because any event of this type is irreversible and seriously interrupts the process of reconfiguration. Despite the importance of this problem for the operation of densely-packed modular robots, but also because of the difficulties involved, the topic has scarcely been pursued and is far from having been solved.

Prediction of mechanical failure requires performing a mechanical analysis of the reconfigured version of a modular robot before the reconfiguration actually occurs, using the methods of computational mechanics [12]. There are two ingredients of this process. The first one is building an adequate mechanical model of the modular robot itself and of the way it is supported and loaded. In different contexts, models of this kind have been investigated in [13], [14] and [15]; generally, many Finite Element approximations can be used to serve this purpose. The second one is an effective

[1]Paweł Hołobut and Jakub Lengiewicz are with the Institute of Fundamental Technological Research, Polish Academy of Sciences, Poland pholob@ippt.pan.pl, jleng@ippt.pan.pl

[2]Stéphane Bordas and Jakub Lengiewicz are with the Department of Engineering, Faculty of Science, Technology and Medicine, University of Luxembourg, Luxembourg stephane.bordas@alum.northwestern.edu, jakub.lengiewicz@uni.lu

numerical solution of the resulting mathematical problem in order to compute the stresses in the system and the robot-environment reaction forces [16]. For the robot to be entirely autonomous, the computations need to be executed by the modules themselves, exploiting their limited memory, computing power, and local inter-modular communication. In other words, the solution algorithm has to run on the distributed asynchronous computing architecture represented by the interconnected ensemble of modules.

In the present paper, the approach to mechanical failure prediction proposed in [12] is significantly improved. The Finite Element representation of the modular robot with linear-elastic beams modeling pairs of connected modules is used again, this time in the general 3D setting. Previously, however, the solution to the system of mechanical equilibrium equations resulting from the model was obtained by the weighted Jacobi iterative procedure. It had the advantage of being easily run on the modular robot itself in a fully distributed fashion, but at the cost of slow convergence characteristic of this procedure. Here, a special implementation of the fast *Conjugate Gradient* algorithm [17] is introduced, as an alternative to the weighted Jacobi, which greatly reduces the necessary computation time. The proposed version of the algorithm spreads computations over the modules, exploiting the robot's particular distributed computing architecture.

The discussion is limited to connection breakage only, leaving out the loss-of-stability failure case as it involves considering unilateral contact conditions which are more difficult to handle. The main focus is on cubic modules, despite the fact that the procedure is general and can address different module geometries and connection schemes. For the sake of clarity, only computation of the mechanical response of an *existing* configuration of a robot is presented. Nevertheless, as it was shown before [12] in the context of a different algorithm, the framework can be extended to allow prediction of the mechanical state of a planned (perturbed) configuration. This can be achieved by splitting a reconfiguration step into several stages, such as releasing connections, shifting modules, and forming new connections, each of which is analyzed mechanically using the presented procedure—necessarily augmented to handle modules which change their place and module-to-module contact conditions.

Implementation and verification of the algorithm is performed in the dedicated modular-robot simulator *VisibleSim* [18]. The simulator is compatible with the modular-robotic system *Blinky Blocks* [19], i.e., the code in *VisibleSim* can be run on the real hardware. Hardware tests are, however, outside the scope of the present paper, partially because assembling a thousand-module structure, like the ones in the simulations section, is currently barely feasible.

II. Problem formulation

A. Preliminaries

We consider a robot built of N cubic modules connected face-to-face using magnetic connectors. Each module is equipped with a CPU, memory and can exchange messages with its face-adjacent neighbours, of which there may be up to six. The robot resides in the gravity field and is possibly subjected to some other external loading assumed to be quantitatively known. There are also additional modules attached to the structure that are fixed—considered to be immobile—which prevent the robot from losing stability and falling. The task is to program the ensemble of modules to autonomously compute the stresses caused by the external loading in all inter-modular connections. The algorithm should be relatively efficient and exploit the distributed computational structure of the robot.

B. Linear-elastic beam model of connected modules

Various mechanical models of the modular robot can be adopted, depending on the required precision and acceptable complexity of the representation. Similarly to [12], the standard Finite Element frame model is used below to represent the modular robot. In the model, each module is represented by a single node with six degrees of freedom—three displacements and three rotations—located in the middle of the module. Each pair of connected modules, in turn, is represented by a linear-elastic beam joining the nodes of the two modules.

In a global coordinate system $\mathrm{CS_G}$, the beam model stipulates that

$$\mathbf{f}_{pq} = \mathbf{K}^{11}_{pq}\mathbf{u}_p + \mathbf{K}^{12}_{pq}\mathbf{u}_q\,, \tag{1}$$

where $\mathbf{f}_{pq} = [f_x, f_y, f_z, m_x, m_y, m_z]^{\mathrm{T}}_{pq}$ is the vector of reaction forces and torques acting on the beam q–p at node p ($\cdot^{\mathrm{T}}$ denotes a transpose), $\mathbf{u}_q = [u_x, u_y, u_z, \tau_x, \tau_y, \tau_z]^{\mathrm{T}}_q$ and $\mathbf{u}_p = [u_x, u_y, u_z, \tau_x, \tau_y, \tau_z]^{\mathrm{T}}_p$ are the vectors of displacements and rotations of nodes q and p, respectively, $\mathbf{K}^{11}_{pq} = \hat{\mathbf{R}}_{pq}\mathbf{K}^{11}\hat{\mathbf{R}}^{\mathrm{T}}_{pq}$ and $\mathbf{K}^{12}_{pq} = \hat{\mathbf{R}}_{pq}\mathbf{K}^{12}\hat{\mathbf{R}}^{\mathrm{T}}_{pq}$ are stiffness matrices of the beam,

$$\hat{\mathbf{R}}_{pq} = \begin{pmatrix} \mathbf{R}_{pq} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_{pq} \end{pmatrix}, \tag{2}$$

$$\mathbf{K}^{11} = \frac{E}{L^3}\begin{pmatrix} 12I_x & 0 & 0 & 0 & -6I_xL & 0 \\ 0 & 12I_y & 0 & 6I_yL & 0 & 0 \\ 0 & 0 & AL^2 & 0 & 0 & 0 \\ 0 & 6I_yL & 0 & 4I_yL^2 & 0 & 0 \\ -6I_xL & 0 & 0 & 0 & 4I_xL^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & J_\nu L^2 \end{pmatrix}, \tag{3}$$

$$\mathbf{K}^{12} = \frac{E}{L^3}\begin{pmatrix} -12I_x & 0 & 0 & 0 & -6I_xL & 0 \\ 0 & -12I_y & 0 & 6I_yL & 0 & 0 \\ 0 & 0 & -AL^2 & 0 & 0 & 0 \\ 0 & -6I_yL & 0 & 2I_yL^2 & 0 & 0 \\ 6I_xL & 0 & 0 & 0 & 2I_xL^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -J_\nu L^2 \end{pmatrix}, \tag{4}$$

$\mathbf{R}_{pq}$ is the 3×3 rotation matrix from the local coordinate system CS_{pq}—whose z axis points from node q to node p—to $\mathrm{CS_G}$, $\mathbf{0}$ is the 3×3 zero matrix, while E, L, A, I_x, I_y and J_ν are the elastic modulus, length, cross-sectional area, area moments of inertia in the x and y direction of CS_{pq}, and scaled torsion constant in the z direction of CS_{pq}, respectively (see Tab. I).

symbol	value	description
E	100 MPa	elastic modulus
$L = a$	40 mm	length
$A = a^2$	40×40 mm^2	cross-sectional area
I_x, I_y	213000 mm^4	area moment of inertia of the beam connection in the x and y direction
J_v	138500 mm^4	scaled torsion constant in the z direction
M	0.061 kg	mass of a block
$F_{\mathrm{V}}^{\max}$	12 N	maximal magnetic force of a vertical connection
$F_{\mathrm{L}}^{\max}$	15 N	maximal magnetic force of a lateral connection

TABLE I

GEOMETRIC AND MATERIAL PARAMETERS OF THE MODULES AND THE CONNECTIONS (MOTIVATED BY THE *Blinky Blocks* MODULAR ROBOT [19]).

C. Equilibrium equations of the modular robot

Since all degrees of freedom of the ensemble of modules are assumed to be located at discrete nodes, with three translational and three rotational degrees of freedom per node, the system is in static equilibrium if and only if the sum of forces and the sum of torques acting on each node are zero. This balance of the forces and torques at any node p requires that the sum of reactions from the beams ending in p, as given by Eq. (1), equals the external forces and torques $\mathbf{F}_p^{\mathrm{ext}} = [f_x^{\mathrm{ext}}, f_y^{\mathrm{ext}}, f_z^{\mathrm{ext}}, m_x^{\mathrm{ext}}, m_y^{\mathrm{ext}}, m_z^{\mathrm{ext}}]_p^{\mathrm{T}}$ acting on node p. One obtains the standard Finite Element system of equations:

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

where $p = 1, \ldots, N$, while q runs over the indices of all neighbors of a given module p, including fixed modules, for which $\mathbf{u}_q = \mathbf{0}$. Vectors $\mathbf{F}_p^{\mathrm{ext}}$ and $\mathbf{u}_p$, for all p, are expressed in the same global coordinate system $\mathrm{CS_G}$. The external forces are assumed to be known; in fact, in the simulations we will consider vertical gravity to be the only factor, in which case $\mathbf{F}_p^{\mathrm{ext}} = [0, 0, -M \cdot g, 0, 0, 0]^{\mathrm{T}}$ for every module p, where M is the mass of one module (cf. Tab. I) and g is gravitational acceleration. The unknowns of the system are the 6-vectors of generalized displacements $\mathbf{u}_p$, $p = 1, \ldots, N$.

As it is usually done in Finite Element analysis, Eq. (5) can be also presented in a compact matrix form

$$\mathbf{K}\mathbf{u} = \mathbf{F}^{\mathrm{ext}}, \tag{6}$$

where $\mathbf{K}$ is the assembled $6N \times 6N$ known constant stiffness matrix, $\mathbf{F}^{\mathrm{ext}}$ is the $6N$-vector of known constant external forces and torques, and $\mathbf{u}$ is the $6N$-vector of unknown nodal displacements and rotations, for which the system needs to be solved. $\mathbf{K}$ is symmetric and positive definite, which follows from the linear-elastic properties of the robot under small deformations. In Section III, a distributed technique for solving Eq. (6) by the modular robot itself is described, which is the main topic of the present paper.

D. Condition for overloading of inter-modular connections

Once Eq. (6) is solved for $\mathbf{u}$, which is the subject of the next section, the robot may autonomously check computationally if any of the connections between its modules has reached an unacceptable stress level. Junction forces and torques between any pair of connected modules can be approximated by the forces and torques in the middle of the beam representing this pair. Using Eq. (1) and observing that the distribution of forces and torques is linear along the beam in the adopted model, one obtains approximate junction forces and torques between modules q and p, expressed for convenience in the local coordinate system CS_{pq}, as

$$\begin{aligned}[f_x^{\mathrm{c}}, f_y^{\mathrm{c}}, f_z^{\mathrm{c}}, m_x^{\mathrm{c}}, m_y^{\mathrm{c}}, m_z^{\mathrm{c}}]_{pq}^{\mathrm{T}} &= \frac{1}{2}\hat{\mathbf{R}}_{pq}^{\mathrm{T}}(\mathbf{f}_{pq} - \mathbf{f}_{qp}) = \\ &= \frac{1}{2}\hat{\mathbf{R}}_{pq}^{\mathrm{T}}(\mathbf{K}_{pq}^{11}\mathbf{u}_p + \mathbf{K}_{pq}^{12}\mathbf{u}_q - \mathbf{K}_{qp}^{11}\mathbf{u}_q - \mathbf{K}_{qp}^{12}\mathbf{u}_p),\end{aligned} \tag{7}$$

where $\mathbf{u}_q$ and $\mathbf{u}_p$ come directly from $\mathbf{u}$ and are thus expressed in the global coordinate system $\mathrm{CS_G}$.

The possibility of connection breakage can be checked by comparing the connection forces and torques obtained from Eq. (7) with limit values valid for a given attachment mechanism. In the case of cubic Blinky Blocks, which use magnetic forces to bind adjacent modules together, the condition for a connection to hold might have the form

$$(F^{\max} - f_z^{\mathrm{c}}) \cdot a/2 > \max(|m_x^{\mathrm{c}}|, |m_y^{\mathrm{c}}|), \tag{8}$$

where $F^{\max}$ is the total attraction force between the magnets of the two modules and a is the size of the cube (cf. Tab. I). The above condition accounts for both tension and bending, which strain the connection simultaneously. We do not use conditions for shearing and twisting because Blinky Blocks' connectors are more resistant with respect to these failure modes.

III. DISTRIBUTED CONJUGATE GRADIENT SOLVER

The matrix $\mathbf{K}$ in Eq. (6) is symmetric and positive definite, which allows one to use the Conjugate Gradient solution scheme in its basic form [17], and assures convenient memory- and CPU complexities with respect to the number of modules (see Section III-D). Extensions to more general cases, e.g., non-linear or non-symmetric systems, are possible (see, e.g., [20]), but in general at the cost of higher memory and CPU requirements.

In Section III-A, the original centralized form of the method is presented, while in Section III-B, its extension to the distributed computing architecture embodied by a modular robot is discussed.

A. Centralized (shared memory) formulation of CG

After [17] and [16], the Conjugate Gradient (CG) iterative solution scheme has the following general form. The procedure is initialized with: $\mathbf{u}_0 = \mathbf{0}$, $\mathbf{r}_0 = \mathbf{F}^{\mathrm{ext}}$, $\mathbf{d}_0 = \mathbf{P} \cdot \mathbf{F}^{\mathrm{ext}}$ and $n = 1$, where $\mathbf{P}$ is a preconditioning matrix (explained below). Each CG iteration n consists of the following sub-steps:

1)
$$\alpha = \frac{\mathbf{r}_{n-1}^{\mathrm{T}} \cdot \mathbf{P} \cdot \mathbf{r}_{n-1}}{\mathbf{d}_{n-1}^{\mathrm{T}} \cdot \mathbf{K} \cdot \mathbf{d}_{n-1}},$$

2)
$$\mathbf{u}_n = \mathbf{u}_{n-1} + \alpha \mathbf{d}_{n-1},$$

$$\mathbf{r}_n = \mathbf{r}_{n-1} - \alpha \mathbf{K} \cdot \mathbf{d}_{n-1},$$

	CG	weighted Jacobi
Execution time	$O(N \cdot D) \simeq O(N^{4/3})$	$O(N^2)$
No. of CPU oper. / module	$O(N)$	$O(N^2)$
No. of messages / module	$O(N)$	$O(N^2)$
Memory usage / module	$O(1)$	$O(1)$

TABLE II
COMPLEXITY ASSESSMENTS FOR THE CONJUGATE GRADIENT ALGORITHM AND THE WEIGHTED JACOBI ALGORITHM. N IS THE NUMBER OF MODULES AND D IS THE DEPTH OF THE SPANNING TREE.

3)
$$\beta = \frac{\mathbf{r}_n^{\mathrm{T}} \cdot \mathbf{P} \cdot \mathbf{r}_n}{\mathbf{r}_{n-1}^{\mathrm{T}} \cdot \mathbf{P} \cdot \mathbf{r}_{n-1}},$$

4)
$$\mathbf{d}_n = \mathbf{P} \cdot \mathbf{r}_n + \beta \mathbf{d}_{n-1}.$$

If the relative residual error $\mathbf{r}_n^{\mathrm{T}} \cdot \mathbf{P} \cdot \mathbf{r}_n / \mathbf{r}_0^{\mathrm{T}} \cdot \mathbf{P} \cdot \mathbf{r}_0$ is below a given tolerance level (close to zero) then the algorithm stops. Otherwise, it sets $n := n+1$ and the iteration sub-steps 1-4 are repeated.

The theoretical upper bound on the number of iterations necessary to *converge to the exact solution* is the size of the system, i.e., the number of degrees of freedom (DOF) of the system, which is equal to $6N$. This property alone allows CG to outperform the simplest weighted Jacobi scheme, which is only approximate and for which the required number of iterations scales with the square of the number of DOF of the system, see Table II. The above upper bound for CG is valid for the more general class of Krylov Subspace methods. The effective number of necessary iterations of the CG solver is usually much lower than that bound, and strongly depends on the physical configuration of the robot and on the choice of an appropriate preconditioning matrix $\mathbf{P}$.

In the present paper, two of the simplest possible choices of $\mathbf{P}$ are investigated:

$$\mathbf{P} = \mathbf{I} \qquad \text{and} \qquad \mathbf{P} = \mathrm{diag}(\mathbf{K})^{-1},$$

where $\mathrm{diag}(\mathbf{K})$ is the diagonal part of $\mathbf{K}$. The former case gives a non-preconditioned scheme, which will serve as a reference solution. The latter form is known as the *Jacobi* preconditioner (it should not be mistaken for the weighted Jacobi method mentioned before in this paper), and gives a surprisingly good speed-up of execution, which will be demonstrated in Section IV. Both forms of $\mathbf{P}$ are local, thus being suitable to be directly applied in the distributed version of the algorithm.

B. Distributed version of CG

In a centralized setting (with access to shared memory), the scalar and vector products required by the CG algorithm can be computed directly. Since the modular robot considered in this paper has no shared memory, a special treatment is necessary to distribute and aggregate data, which deteriorates the parallelism of the original CG steps.

The distributed version of the CG algorithm relies on a tree structure (see. Section III-C) that allows efficient distribution and aggregation of information across the robot. We can distinguish three phases of the distributed algorithm, analogous to the centralized version presented in Section III-A. Each phase of the algorithm is initiated (synchronized) by the centroid module which is the root of the spanning tree. If it is not stated otherwise, the information is always propagated/aggregated over the tree. In the notation below, iteration steps n are omitted for simplicity, the lower index p refers to the module number, while q denotes direct neighbors of p.

Phase INIT: Initialization of the CG procedure.

- Each module p initiates its local 6-vectors $\mathbf{u}_p$ and $\mathbf{r}_p$:
$$\mathbf{u}_p = \mathbf{0} \qquad \text{and} \qquad \mathbf{r}_p = \mathbf{F}_p^{\mathrm{ext}}.$$
- The scalar r is aggregated over the tree:
$$r = \sum_p \mathbf{r}_p^{\mathrm{T}} \cdot \mathbf{P}_p \cdot \mathbf{r}_p,$$
where $\mathbf{P}_p$ is the 6×6 central submatrix of $\mathbf{P}$ corresponding to the DOF of module p.
- The centroid sets $r_{\mathrm{ref}} = r$, distributes $\beta = 0$ over the tree to all modules and runs **Phase** α.

Phase α:

- Each module p updates its local 6-vector $\mathbf{d}_p$:
$$\mathbf{d}_p = \mathbf{P}_p \cdot \mathbf{r}_p + \beta \mathbf{d}_p,$$
and exchanges $\mathbf{d}_p$ with all its neighbors q.
- Each module p computes its local 6-vector $\mathbf{z}_p$:
$$\mathbf{z}_p = \sum_q \mathbf{K}_{pq}^{11} \cdot \mathbf{d}_p + \mathbf{K}_{pq}^{12} \cdot \mathbf{d}_q.$$
- The denominator of α is aggregated over the tree:
$$a = \sum_p \mathbf{d}_p \cdot \mathbf{z}_p.$$
- The centroid computes $\alpha = r/a$, distributes it over the tree to all modules and runs **Phase** β.

Phase β:

- Each module p updates its local $\mathbf{u}_p$ and $\mathbf{r}_p$:
$$\mathbf{u}_p = \mathbf{u}_p + \alpha \mathbf{d}_p,$$
$$\mathbf{r}_p = \mathbf{r}_p - \alpha \mathbf{z}_p.$$
- The numerator of β is aggregated over the tree:
$$b = \sum_p \mathbf{r}_p^{\mathrm{T}} \cdot \mathbf{P}_p \cdot \mathbf{r}_p.$$
- The centroid computes $\beta = b/r$ and sets $r = b$.
- The centroid checks if $r/r_{\mathrm{ref}} \simeq 0$. If so, convergence has been achieved and it stops the algorithm. Otherwise, it distributes β over the tree to all modules and runs **Phase** α again.

C. Centroid and spanning tree

The module assigned as the root of the spanning tree should lie close to the center of the robot (center of the connectivity graph of the robot, not necessarily its geometrical center). This reduces the depth of the tree, thereby also the execution time of the algorithm (see Section III-D). The selection of a centroid can be made in a distributed manner, see e.g. [21], but in the present implementation we make that selection manually.

In a distributed and asynchronous architecture, the well-known sequential breadth-first search algorithm for building a spanning tree needs an additional synchronization phase to assure that the depth of the tree is minimal. For a tree of depth D, the number of sequential steps of the algorithm is of the order $O(D^2)$ because the creation of each level of the tree requires a synchronization phase.

D. Complexities of the distributed CG solver

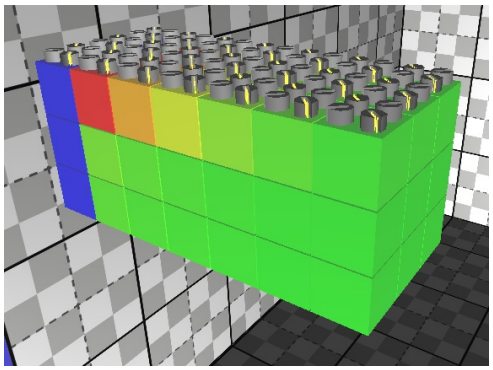
Assessments of the time-, CPU-, communication- and memory complexities of the CG algorithm are presented in Table II, in comparison to the weighted Jacobi solver from our previous paper [22]. In the table, N is the number of modules, D is the depth of the spanning tree. D is of the same order of magnitude as the radius of the graph representing the connection topology of the robot. In the most likely (favorable) scenarios, $D \sim N^{1/3}$ for three-dimensional structures, but in the worst-case scenario $D \sim N$, which can destroy the good scalability properties of the proposed CG algorithm.

In Table II, the execution time is understood as the number of sequential operations of the algorithm: the number of global iterations of the CG algorithm multiplied by the number of the propagation/aggregation steps that are done sequentially level by level over the spanning tree. In the complexity assessment we assumed the rough upper bound N for the number of global CG iterations. In the numerical examples (see, e.g., Fig. 4), however, it can be seen that the number of global iterations grows slower than linearly with the system size (which is favorable). One of possible explanations is that we increase the resolution of robot geometries with almost no increase in their topological complexities, which can be relatively more easy to solve by the CG algorithm.

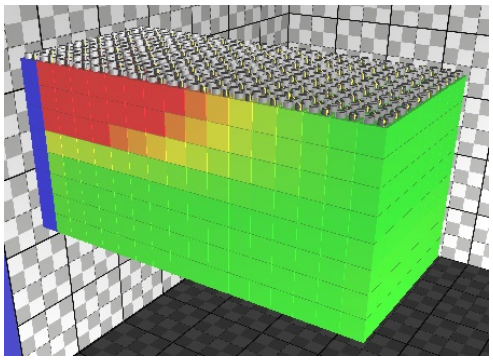
At a given global iteration of the CG algorithm, the maximum numbers of CPU operations and message exchanges are constant per module (although not performed in parallel by all modules). Therefore, in both cases the complexities are $O(N)$. The memory usage per module is only related to the number of direct neighbours, which is constant and does not depend on the size of the system.

IV. Implementation and simulation results

The algorithm has been implemented in the *VisibleSim* emulator [18]. The information exchange between modules is performed solely by sending messages between direct neighbors through message-passing interfaces. The implementation is entirely compatible with the *Blinky Blocks* reconfigurable modular robot [19].



(a) Low resolution: 54 modules, 324 DOF.



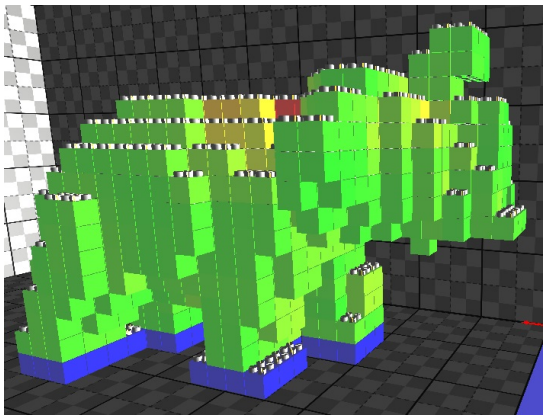
(b) High resolution: 1024 modules, 6144 DOF.

Fig. 1. Cantilever example. Colors represent the level of overloading of connections: green=low, yellow–orange=moderate, red=overloaded. Modules in blue are fixed.

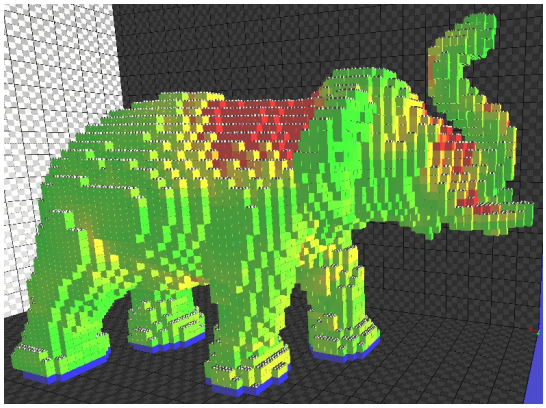
The performance of the algorithm has been analyzed in two types of setups: (1) the cantilever examples, see. Fig. 1, and (2) the elephant examples, see Fig. 2. In both cases we examined the convergence and scaling properties by running the algorithm on setups of growing sizes (resolutions).

The first observation is that, as expected, the Conjugate Gradient method converges to the numerical zero, see Fig. 3. Moreover, the convergence rate increases while getting closer to the solution, which is a characteristic behavior of CG. The second observation is that the simple Jacobi preconditioning significantly improves the convergence of the CG algorithm compared with the non-preconditioned version. For instance, in the case presented in Fig. 3, the preconditioned version required over eight times less global CG iterations to converge. That improvement seems to grow with the growing system size, which is shown in Fig. 4.

In Fig. 4 it is analyzed how the computation time scales with the increasing number of modules. Two measures of computation time are presented: the necessary number of global CG iterations (Fig. 4a) and the necessary number of subsequent CG steps (Fig. 4b). The former measure does not include the propagation/aggregation phases that involve the spanning tree, which refers to a hypothetical situation when the modules have access to shared memory and are able to



(a) Low resolution: 1003 modules, 6018 DOF.



(b) High resolution: 20212 modules, 121272 DOF.

Fig. 2. Elephant example. Colors represent the level of overloading of connections: green=low, yellow–orange=moderate, red=overloaded. Modules in blue are fixed.

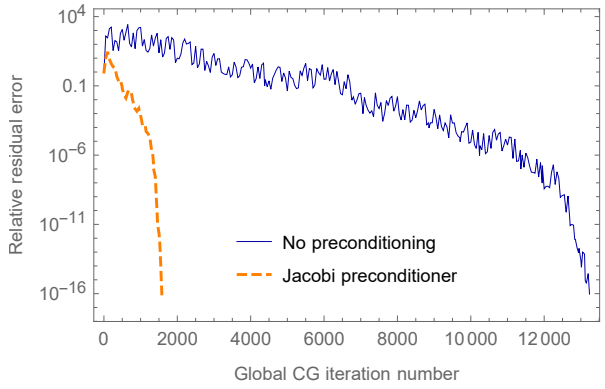
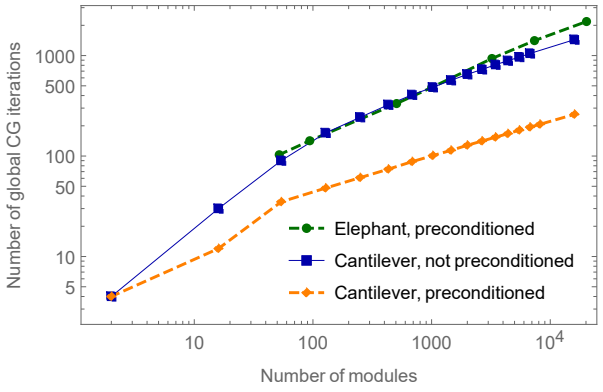


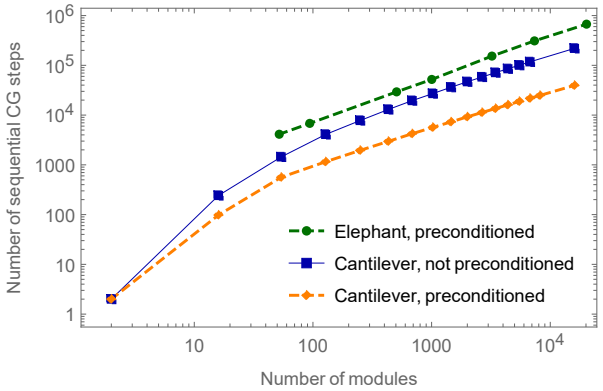
Fig. 3. Elephant example with 7360 modules and 44160 DOF. Convergence of global CG iterations with and without preconditioning.

compute the necessary scalar/vector product in parallel in one step. The latter measure refers to the fully distributed algorithm analyzed in the present paper.

In Fig. 4b one can observe that for bigger configurations the computation time scales with the number of modules linearly or even better (e.g., for 10 times larger problems the convergence time increases no more than 10 times). This result is better than the assessments provided in Table II, as has been discussed in Section III-D. We can also observe that for the same system sizes the Elephant examples converge slower than the Cantilever examples. This is due to the higher geometrical complexity of the Elephant's shape which results in a more complex mechanical response.



(a)



(b)

Fig. 4. Scaling of the number of (a) global CG iterations and (b) sequential CG steps for the cantilever and elephant tests.

V. Conclusions and Future Work

The problem of determining the mechanical state of a modular robot was posed in the form of linear force-balance equations resulting from a simple mechanical model of the robot, together with simplified criteria for inter-modular connection overloading. To solve the system of equations, a version of the Conjugate Gradient method was proposed that can be run by a modular robot in a fully distributed fashion: using distributed CPU and memory, and local communication. The time-, CPU-, communication- and memory complexities of the algorithm were evaluated and proved superior to those of the simple weighted Jacobi scheme. The algorithm was implemented in the VisibleSim emulator and its performance checked in two series of examples. The computation time was observed to grow slower than linearly with the system size.

The proposed approach complements our research into building a scalable framework for modular-robotic Programmable Matter. In general, reconfiguration time and system strength tend to deteriorate when the number of modules increases and their dimensions decrease. So far, we have

demonstrated efficient schemes allowing high parallelism of reconfiguration, while assuring structural integrity of a robot [10]. We have also presented a family of modular-robotic actuator structures which produce forces proportional to the number of their modules [22]. However, a reconfiguration planner is still missing that would efficiently predict the future mechanical state of a robot. The presented algorithm is a step towards breaking that efficiency barrier.

The performance of the present algorithm is much better than that of the weighted Jacobi method. However, the number of necessary operations seems to be still too high for large-scale systems. There are several possibilities for further improvement of the efficiency of the framework:

1) Multigrid techniques [23], which form a family of more accurate preconditioners and can thus reduce the necessary number of iterations of the solver.
2) Model order reduction techniques, which could simplify the mechanical model and thus reduce the size of the system to be solved. In particular, methods capable of creating connections between proper orthogonal decomposition and Newton-Krylov methods [24] would be particularly attractive in terms of resolving non-linearities and localization. Those in turn can be generalized using selective reduced domain decomposition methods, as in [25]. These kinds of methods have already been applied in soft robotics [26], albeit, in a centralized setting.

There are several other possible extensions of the present work. (1) In addition to the overload check, a stability check is necessary to fully evaluate the mechanical state of a robot. This is more involved because unilateral contact conditions must be included, introducing non-linearities and possible asymmetry to the system of equations. Special solution techniques are therefore required. (2) Development of a more accurate mechanical model of the modular robot itself. (3) Application to different connection topologies, in particular, to quasi-spherical module designs [27]. (4) Running the algorithm on real *Blinky Blocks* [19], which would allow experimental validation of the framework.

ACKNOWLEDGMENT

This work was partially supported by the EU Horizon 2020 Marie Sklodowska Curie Individual Fellowship *MOrPhEM* (H2020-MSCA-IF-2017, project no. 800150)

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

[3] 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.

[4] K. Støy, *Emergent control of self-reconfigurable robots*. PhD thesis, The Maersk Mc-Kinney Moller Institute for Production Technology, University of Southern Denmark, Odense, Denmark, 2004.

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

[6] M. De Rosa, S. C. Goldstein, P. Lee, J. Campbell, and P. Pillai, "Scalable shape sculpting via hole motion: motion planning in lattice-constrained modular robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1462–1468, 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] Z. Butler and D. Rus, "Distributed planning and control for modular robots with unit-compressible modules," *International Journal of Robotics Research*, vol. 22, no. 9, pp. 699–715, 2003.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] J. Hiller and H. Lipson, "Automatic design and manufacture of soft robots," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 457–466, 2012.

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

[16] L. N. Trefethen, *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, jun 1997.

[17] M. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Journal of Research of the National Bureau of Standards*, vol. 49, p. 409, Dec. 1952.

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

[19] 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.

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

[21] 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.

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

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

[24] 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.

[25] P. Kerfriden, O. Goury, T. Rabczuk, and S. P.-A. Bordas, "A partitioned model order reduction approach to rationalise computational expenses in nonlinear fracture mechanics," *Computer methods in applied mechanics and engineering*, vol. 256, pp. 169–188, 2013.

[26] O. Goury and C. Duriez, "Fast, generic, and reliable control and simulation of soft robots using model order reduction," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1565–1576, 2018.

[27] 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.