DisCo: A Multiagent 3D Coordinate System for Lattice Based Modular Self-Reconfigurable Robots

Benoît Piranda Frédéric Lassabe Julien Bourgeois

***Abstract*— Localizing each module in a modular self-reconfigurable robot (MSR) is of paramount importance. In MSR, the communication graph is directly mapped to the real topology which makes the localization problem easy to solve. However, some types of connectors can lose the orientation of the modules, making the problem intractable. In this work, we propose to build a coordinate system for 3D lattice-based modular robots using a multiagent system. We present *DisCo* algorithm, that uses one agent per module which can only communicate with its connected neighbors and that does not need a central coordination system. We show that the agents can tackle any kinds of 3D lattice and we illustrate it with a Face Centered Cubic lattice (12 neighbors) and a cubic lattice (6 neighbors). Using communications and only four states, *DisCo* can also deduce the orientation of modules if the connectors do not provide this information.**

I. Introduction

Building a robot from a set of interconnected modules has been an active research field for more than forty years now. Each module forming such a modular robot can also be enhanced by movement capabilities allowing reconfiguration of the modules in space. Such kind of systems have been called metamorphic robotic system and later on modular self-reconfigurable robots (MSR). Lattice-based MSR are one category of MSR in which the modules are aligned on a regular lattice. To ensure scalability, these systems use neighbor-to-neighbor communication such that adding one more module cannot overload the network capability compared to a bus. From a logical point of view, each module can be seen as an agent communicating with its neighbors and forming a multiagent system. An interesting characteristics of MSR is that the network connection which is a logical information, can also be interpreted as a physical information: the location of each neighboring module.

This location of the connected neighbors is a crucial information as it can be used to derive the shape of the ensemble of robots. Building a global coordinate reference system which can be applied to any kind of robots, consists in locating the position of all modules in the lattice and identifying their orientation.

If every module can have the relative location of its neighbors and their orientation, the problem of building the coordinate system is quite straightforward but it can be impossible if the orientation information is missing. Figure 1 shows a set of *Blinky Blocks* [11], which are robots that can deduce the orientation of robots placed on the same plane but cannot get vertical orientation of their top and bottom neighbors. On the left picture, the reference robot is the rightmost *Blinky Block* and it is able to communicate the relative coordinates and orientations to all green robots. Yellow robots on the second floor and above can be placed but not oriented. The orientation and location of blue *Blinky Blocks* could not normally be retrieved as they are connected to yellow robots which do not know their orientation. However, there is an algorithmic solution to retrieve their position and orientation. The red *Blinky Block* can not be placed and oriented at all and it is therefore not possible to differentiate the 4 red robots connected to the top yellow one (cf. Figure 1b). This can cause a collision when moving without a sensor or can cause an incorrect representation of the global shape of the robot.

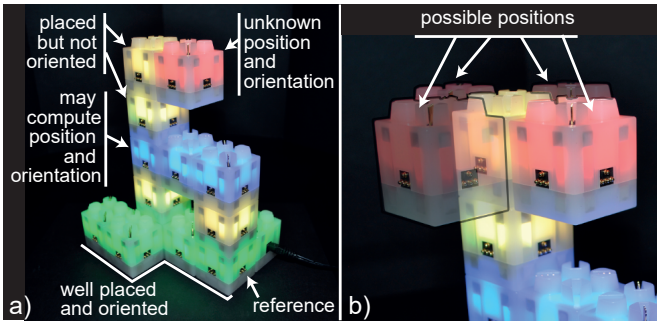

Fig. 1. Illustration of the issue of the coordinates definition on *Blinky Blocks* robots.(Better printed in color)

In this paper, we propose *DisCo*, a robust multi-agent architecture that builds a 3D coordinate system for lattice-based modular robots. This algorithm uses one agent per module which is able to communicate only with its direct connected neighbors with no central coordination system. We demonstrate that the agents can tackle any kinds of 3D lattice even if the relative orientations of neighbors is not strongly defined. We show experiments with a face centered cubic lattice (the *3D Catoms* [19], 12 neighbors) and a cubic lattice (the *Blinky Blocks* [11], 6 neighbors). Using communications and only four states, they can also deduce the orientation of modules if the connectors do not give this information.

In the remainder of this paper, we study existing localization algorithms for MSR and swarm. Second, we propose a global generic positioning model which aims at solving the positioning problem for any robot model (2D, 3D, mobile, static). Third, we estimate our model performances by implementing it on specific MSR platforms, through simulations

This work was partially supported by the ANR (ANR-21-CE33-0009) and the EIPHI Graduate School (ANR-17-EURE-0002).

University of Franche-Comté, FEMTO-ST institute, CNRS, Montbéliard, France [name].[surname]@femto-st.fr

with *VisibleSim* [26], and through an implementation on real hardware, the *Blinky Blocks*.

II. Related work

The first thing to localize modules into an ensemble is to define a coordinate system which adapts to the different kind of lattice. There are many of them like Bravais lattices [3] but the most used in robotics is the Cartesian coordinate system augmented with the orientation information. In swarm robotics, modules are not aligned on a lattice and most of the time the space is a 2D space, simplifying the problem compared to a 3D one. Robots use wireless communications (infrared, ZigBee, etc.) which are used to evaluate the distance between them and build an approximated coordinate system. In [14], McLurkin defines the basic algorithms for sensor networks to determine physical positions from network topology: distance estimation, error estimation, neighbor counts and edge detection to cite a few. In the amorphous computing project, first ideas of a 2D coordinate system for sensor networks is found in [1], [5], [15] using communication propagated waves and three anchors and it is refined in [16]. Minimizing communications while having a sufficient accuracy is a complicated trade-off. In [12], the authors propose to use co-variance intersection [10] with moving robots.

Although there are similarities between swarm computing localization algorithms and modular robots coordinate systems, the main difference lies in the fact that the communication graph in modular robots is directly mapped to the real topology. Chirikjian proposed in [4] the first coordinate system for 2D hexagonal lattice without orientation. A different and more detailed method is proposed in [22] to localize robots in 2D and 3D for hexagonal lattices but orientation works only for peripheral nodes as it needs movement from the robots to reach a consistent orientation. In [2], the authors use a coordinate system in 3D without orientation to build self-assemble shapes. When the number of modules and the imprecision of lattice increase, a precise localization is difficult to achieve. In [6], the authors propose an approximated 3D localization based on probabilistic evaluation of localization using a Maximum Likelihood Estimate (MLE) and a partition of the ensemble between dense areas and sparse ones. The results don't give a precise coordinate system but rather a rough estimate of the location, especially in sparse areas. In [9], the authors propose an algorithm for localizing modules in a 2D grid, without needing the orientation of the modules. This is the closest work to what we propose and the main differences are that they only consider a single topology and moreover only in 2D. Furthermore, the algorithm is synchronous, meaning that it needs a global clock or orchestration, and global halting condition has not been implemented.

III. The *DisCo* multi-agent architecture

A. Theoretical model

In this work, we consider lattice-based MSRs with robots communicating with their connected neighbors using connectors disposed on the surface of the robot.

In a first approach, we consider that we can express the position and the orientation of the neighbor connectors relatively to the local coordinate system of the robot. Under this hypothesis, when a robot is connected to a neighbor, it can get the relative orientation of this neighbor. This constraint is very strong but can be solved by the latching system. We will show in Section III-B that it is possible to relax it using the network of robots.

In this article, we consider robots with 6 to 12 connectors in 3D - which is the maximum number that can be reached on spherical robots aligned in a Face Centered Cubic lattice (FCC) lattice. Specialized robots may for instance have 12 connectors for *3D Catoms* [19] or *Datoms* [20], 6 connectors cubic *Blinky Blocks* [11], Roombots [25], MBlocks [23], Pebbles [7] or Miche [8], 6 connectors for 2D hexagonal robots (like mm catoms [18]), 4 for Crystalline [24] or Smart blocks [21] in square lattice. Our method is also applicable to heterogeneous robots combining several shapes of robots in a same set. The only constraint is the position and orientation of each connector in the local coordinate system of the robot.

1) The robot model: Our goal is to deduce the (x_i, y_i, z_i) coordinates for every robot of a connected ensemble ($i \in [0..n]$). Although most of the robots are placed in a 3D environment, our model can process 2D environments by simply assuming the z coordinate to be 0. Only knowing its 3D coordinate is not sufficient enough for a robot to be part of a global coordinate system. It must also know its orientation defined by its angles related to the $(\vec{X}, \vec{Y}, \vec{Z})$ vectors of the global coordinate system. Then, each position and orientation of a robot k may be described by an homogeneous transformation matrix M_k.
For each robot, we consider a list of connectors C_i ($i \in [0..m]$) which are placed and oriented in the local coordinate system of the robot applying a matrix W_i.

2) Computing the position of a neighbor: If we consider a robot A, already placed in a referential, and connected to another robot B using the connectors C_i from A and C_j from B, then, the position and orientation of this second robot is given by the following relation:

$$M_A \cdot W_i \cdot S = M_B \cdot W_j \tag{1}$$

Where S is the symmetry operator which must be applied to C_i to get the orientation of C_j. We can express M_B from M_A in the general case by:

$$M_B = M_A \cdot W_i \cdot S \cdot (W_j)^{-1} \tag{2}$$

Equation 2 shows that robot B can deduce its position and orientation using the matrix $M_A \cdot W_i$ received from A.
We can then deduce the following algorithm:

- First, elect the *Coordinate System Origin* (*CSO*), which set its local matrix M_{CSO} to the identity and flood $(M_{CSO} \cdot W_i)$ to all its direct neighbors.
- Then, when a module B receives the matrix for the first time, it computes its position and orientation M_B applying Equation 2, and sends the $(M_B \cdot W_i)$ to all its neighbors but the sender.

B. Real case, an algorithm to relax constraints

When a robot is connected to a neighbor, the neighbor fill one of the cell of the associated lattice, the position of the neighbor can be deduced with confidence but its orientation around the connector is not always known.

In the following, we present *DisCo*, a full distributed algorithm to deduce the position and orientation of robots even if connection information are partial.

We first use the *Blinky Block* robot shape, *Blinky Blocks* are not fully symmetrical robots (see Figure 2 and 1), these Lego like robots are placed vertically on the floor or over a neighbor, but rotations around the vertical axis cannot be detected by the hardware. As a consequence, we must consider two kinds of connectors: lateral connectors $\{East, North, West, South\}$ sharing the horizontal plane and the $\{top, bottom\}$ connectors. For the lateral connectors, only one orientation of connection is possible (the two neighbors must be vertically aligned), then all robots placed in the same horizontal plane of an already placed and oriented robot are placed and oriented also as presented in section III-A .

But for neighbors placed on the top face (or the bottom face), there are 4 possible rotations around the vertical axis. Then a *Matrices* message to a neighbor placed on one of its $\{Top, Bottom\}$ faces will embed 4 matrices.

On *Blinky Block* robots, the $\{East, North, West, South\}$ connectors C_i (with $i \in [0..3]$) are placed at the center of the lateral faces. We express W_i by the following homogeneous transformation matrices:

$$W_i = \begin{pmatrix} \cos\left(\frac{i\pi}{2}\right) & -\sin\left(\frac{i\pi}{2}\right) & 0 & \frac{1}{2}\cos\left(\frac{i\pi}{2}\right) \\ \sin\left(\frac{i\pi}{2}\right) & \cos\left(\frac{i\pi}{2}\right) & 0 & \frac{1}{2}\sin\left(\frac{i\pi}{2}\right) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (3)$$

And for the $upper$ and $lower$ faces:

$$W_{Upper} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4)$$

and

$$W_{Lower} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & -\frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (5)$$

For lateral connectors the symmetry matrix is unique and equal to:

$$S_{lateral} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (6)$$

But for top and bottom faces, the symmetry matrix may be one of the 4 following matrices S_i with $i \in [0..3]$:

$$S_i = \begin{pmatrix} \cos\left(i\frac{\pi}{2}\right) & -\sin\left(i\frac{\pi}{2}\right) & 0 & 0 \\ \sin\left(i\frac{\pi}{2}\right) & \cos\left(i\frac{\pi}{2}\right) & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (7)$$

We can notice that in the case of *Blinky Blocks*, when several modules are placed one on top of the other, the combinations of matrices can be simplified to obtain only 4 different matrices. But of course, the number of stored matrices can become larger when modules are connected on the side faces of these *Blinky Blocks*.

1) Orientation Resolution algorithm: The model starts by choosing the *CSO*, which can be arbitrarily set (based on an identifier value for instance), or elected with some constraints in terms of connectivity. This *CSO*'s orientation is set to be the global orientation.

Each agent placed on each robot stores a list of candidate matrices encoding potentials positions and orientations. When a robot floods geometrical data, it sends the list of all possible matrices to a connected neighbor. When the neighbor receives this list, it merges it with its local current list in order to add new possible matrices or to reduce the list by confirming possibilities and/or by deleting others.
Each agent can take four different states, associated with the size of the list of matrices.

- UNKNOWN is the initial state of the agents except the *CSO* which is in PLACED AND ORIENTED state, it means the agent has never received any matrix. This state corresponds to an empty list of matrices.
- UNTRUSTED: More than one possible position have been received. This situation appears when the list contains more than one matrix and the matrices do not correspond to the same position.
- PLACED ONLY: The list of matrices contains many matrices but only the orientation differs.
- PLACED AND ORIENTED: The list of matrices is reduced to only one matrix which gives a sure position and orientation of the robot.

As soon as an agent receives a `Matrices Message` it goes from UNKNOWN to one of the three other states

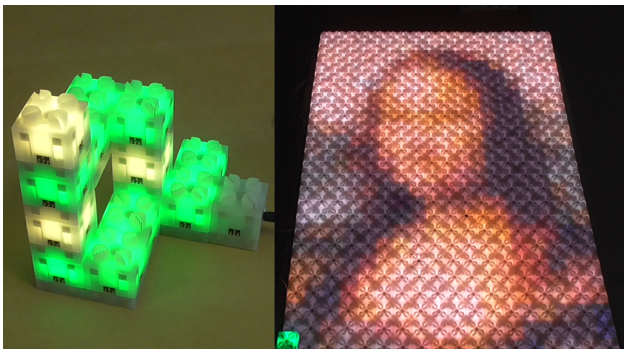

Fig. 2. Two pictures of experiments with *Blinky Blocks* extracted from the video.(Better printed in color)

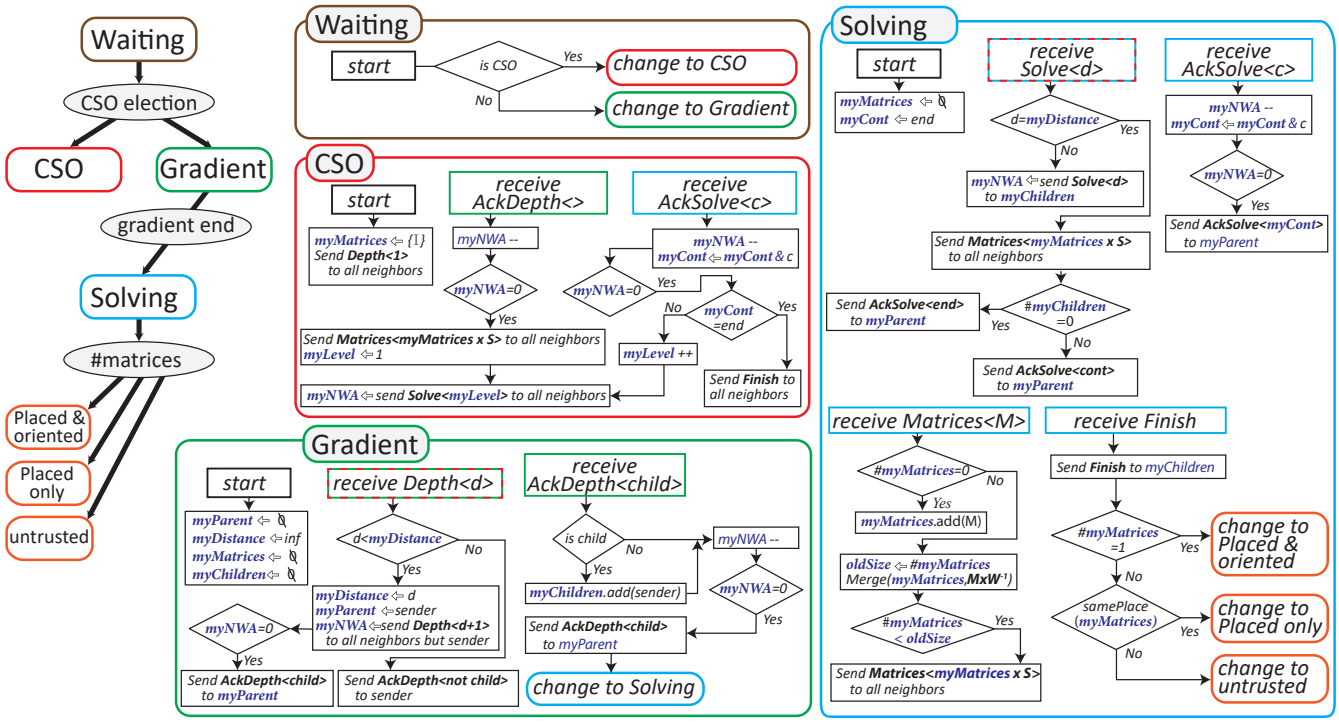

Fig. 3. On the left: the global graph of agents executed in the robots during the algorithm with transition events. On the right, the *DisCo* algorithm detailed by agent.

depending on the size of the sent list of matrices.

In order to change their state, agents must receive a message that embeds a list of matrices. Merging these matrices with their local matrix reduces the number of possible matrices. The `merge` function compares the current list of matrices and the one received in the message in order to retain only compliant ones. If the receiving robot has an empty list, it copies all matrices from the source. Else, it retains only the intersection of both matrices lists.

2) Messages and agent attributes: Each agent stores the list of matrices in `myMatrices`. In our implementation, to reduce the memory size of embedded variables, we compress each matrix in 7 bytes (6 bytes for the coordinates and one for the orientation, coded by the number of the connector aligned with $\overrightarrow{x}$).

3) Breadth first search algorithm: The main problem of the algorithm is its termination, indeed an agent cannot know when the process of matrices exchange is finished. Only agents in PLACED AND ORIENTED state have finished the process but all the others may wait for a new matrix forever.

To tackle this problem, we propose to use a Breadth First Search (BFS) that activates step by step the diffusion of the matrices to all children with the same distance to the root module. When the farthest modules have finished their diffusion and return an acknowledgment to the *CSO*, the positioning process is finished.

The *DisCo* algorithm requires three main steps and 4 agent states as shown in the right side of Figure 3. Initially, all agents are in the "**Waiting**" state:

1) The first treatment is the election of the *CSO*, the elected module changes to "**CSO**" state and the others go to the "**Gradient**" one.
2) The second step consists in creating a spanning tree from the *CSO*, storing at each node the list of children ($myChildren$ attribute) and the distance to the *CSO* ($myDistance$ attribute), which corresponds to the depth of the node in the tree. At the end of the process, modules send an acknowledgment to the *CSO*, and change to the "**Solving**" state. Steps 1 and 2 may be merged into one step when using an election algorithm such as ABC-Center [17].
3) During the third step, the *CSO* agent sends its matrices to all its children and activates all the nodes at the same distance from itself and wait for an answer. It activates all modules at distance 1 first, then modules at distance 2, and so on. When activated, an agent sends its matrices to all its neighbors. These matrices are merged with the others in order to reduce the list of possible positions. If the number of matrices is reduced, it re-sends the new list to its neighbors.
4) Finally, the *CSO* broadcasts a message such that each module will have its final state: "PLACED AND ORIENTED ", "PLACED ONLY " or "UNTRUSTED ". Each module is then ready to begin the execution of another application which would need the coordinates and orientations of the modules.

For more details, this algorithm uses 6 different messages presented in the right part of the Figure 3. `Depth` and `AckDepth` are used for building the spanning tree and inform the *CSO* at the end of this step. `Solve` and `AckSolve`

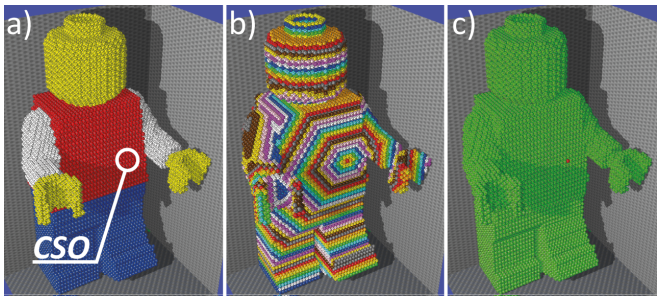

Fig. 4. 3 steps of the algorithm illustrated on the LegoGuy model (45,189 modules) using *VisibleSim* [26].
a) Election of the *CSO* module. b) Gradient from the *CSO*, the colors represent the distance to the *CSO*. c) Merging of matrices. (Better printed in color)

are needed for activating nodes and for informing the *CSO* at the end of the activation. The message `Matrices` is sent by activated nodes to exchange matrices between neighbor modules. Finally, the `Finish` message is in charge of the termination of the algorithm.

4) Generalization to 3D Catoms geometry and connections: The *3D Catom* is a quasi-spherical module placed in a FCC lattice and connected to up to 12 neighbors. Considering two *3D Catoms*, latched using two connectors, there are two possible orientations called *up* and *down* as shown in Figure 5. The IDs of a connector are written in black for the *up* orientation and in red for the *down* orientation, the axis represent the local coordinate frames. Figure 5b shows the two possible orientations of a latched *3D Catom* for the same couple of connectors. The gray *3D Catom* (left) is latched using its connector #1, and the right *3D Catom* uses the same connector labeled #6 on the top (yellow) and #18 on the bottom (blue) depending on its orientation, *up* (top, yellow) and *down* (bottom, blue). As a consequence, each *3D Catom* placed in the FCC lattice can be oriented in 24 different ways. To implement the computation of neighbor position on *3D Catoms*, we define the twelve W_i matrices ($i \in [0..11]$) deduced from the coordinate systems of the connectors C_i. And we consider two symmetry matrices S_{up} and S_{down} which corresponds respectively to a rotation of 180° around the $\overrightarrow{z}$ and the $\overrightarrow{y}$.

We implement *DisCo* in *VisibleSim* simulator to show the

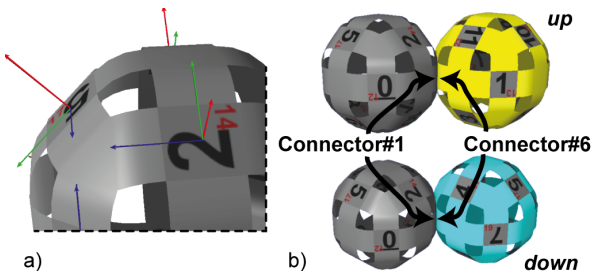

Fig. 5. a) Zoom on a *3D Catom* connector with its IDs: black for the *'up'* orientation and red for the *'down'* orientation. b) Two *3D Catoms* linked by the same connectors #1 and #6. On the top, right *3D Catoms* (in yellow) is oriented *'up'*, on the bottom, right *3D Catoms* (in blue) is oriented *'down'*.

different steps of the *DisCo* algorithm. Figure 4a presents our model made of 45,189 *3D Catoms* randomly oriented, and the position of the elected *CSO*. In Figure 4b, the colors represent the distance to the *CSO* after the gradient step. With this configuration all modules are well placed and oriented by the *DisCo* algorithm as it can be seen from the green color of the Figure 4c.

5) Time complexity of the algorithm: As the computation time is negligible in regards of the communication time, the complexity of the "Gradient" and the "Solving" steps is expressed in function of the time of sending and receiving a message between two neighbors, named t_c.

The "Gradient" step consists in constructing a spanning tree from the *CSO* module whose complexity is $O(d \times t_c)$ [13]. The "Solving" step consists in activating sequentially all the modules at a distance $r = k$, with $k \in [1..\epsilon]$. The complexity of this step is therefore $O\left(2 \times \sum_{r=1}^{\epsilon} r\right) = O(\epsilon^2)$.
Whatever the position of the *CSO* in the configuration, we have $\epsilon \leq d$ where d is the ensemble diameter. Therefore the time complexity of the whole algorithm is $O((d+\epsilon^2)t_c) \approx O(d^2 \times t_c)$.

This result can be checked in Figure 7, drawing the simulation time in number of messages produced during the simulation of the *DisCo* algorithm depending on the eccentricity. It shows two graphs for two different models made of a wide number of modules (up to 45,000) and an eccentricity from 10 to 111. The two curves are aligned, which shows that the time of the simulation does not depend on the shape (nor the number of modules) of the configurations but on the eccentricity of the *CSO* only.

IV. Experiments and Results

The algorithm has been implemented both in real robots *Blinky Blocks* and in the *VisibleSim* simulator. This last tool allows to scale up the number of modules. In Figures 7 and 8, we use *VisibleSim* to compare the duration times for two different shapes: a cube and the *LegoGuy* presented Figure 4. These two objects being vectorial (CSG object as presented in [27]), it is easy to increase the number of modules in resizing the objects. We simulate 25 configurations made of up to 46,656 *3D Catoms*.

We observe in Figure 8 that the shape of the configuration has an effect on the simulation time for the same number of modules. The cube being a dense object, other shapes will have a larger diameter for the same number of modules, which will produce a longer time of computation.

About memory used in modules during the algorithm, it needs a fixed number of local variables (`myDistance`, `myParent`, `myChildren`, etc.), plus the list of matrices. The matrices are encoded in modules using 4 integer (3 for coordinates and one for the orientation). The number of matrices depends on the size of "one module large" loops. When the ensemble is dense, for example in the shape of Figure 4, the maximum number of matrices stored in a agent is 4. But in the disadvantageous case presented Figure 6

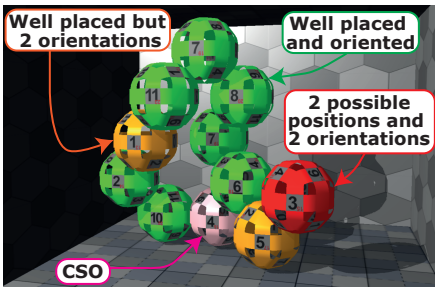

Fig. 6. Worst case configuration of *3D Catoms*. Green modules are well placed and well oriented. Orange one are well placed but many orientations are possible. The position and orientation of the red module cannot be deduced by the algorithm.(Better printed in color)

made of a thin line of 9 *3D Catoms*, the max number of stored matrices reaches 16. In this example, 8 out of 11 modules are well placed and well oriented (in green or pink), 2 are well placed but 2 orientations are possible for each of them (in orange), and the right isolated red module can have 2 possible positions and orientations.

The video[1] shows the algorithm in action for the two implementations. First in *VisibleSim*, we can see the propagation of several messages in large sets of modules (up to 45,000). Then, we present many different executions of the algorithm implemented on 17 to 450 *Blinky Blocks*. We also show that the algorithm reacts to dynamical updates of the configuration by adding and removing modules to a set of modules that already have their coordinates.

V. Conclusion

When a latching connector in a lattice based MSR does not give the orientation of the connected module, setting up a coordinate system is not trivial. In this paper, we propose a multiagent system which uses communication to solve this loss of orientation in the general case of 3D lattice based MSR, no matter the kind of lattice being used. The complexity of the algorithm is in $O(d^2)$, d being the diameter of the system. We test this algorithm in two different contexts, a simulator, to show the scalability and a real modular robot, to prove our algorithm works in real conditions. We show our method scales up to 45,000 robots in simulation and using 450 real *Blinky Blocks*, we validated the dynamical use of *DisCo*. Indeed, *Blinky Blocks* are connected/disconnected during the execution of the algorithm without causing any issue.

We envision several future works to enhance *DisCo* and different applications that would benefit from a coordinate system on MSR.

First, we observed that it is possible to enhance the algorithm by adding a validation process that uses a larger neighborhood. It would reduce the number of matrices by checking if a possible orientation of a robot induces collisions or connections that does not exist.

Then, the choice of the *CSO* could be enhanced by choosing a central node, however, there is a trade-off to be studied in the benefit of running a center election algorithm, which will also use resources.

Many uses stem from having a consistent coordinate system over a MSR, like for example, *Fault detection* and *ID assignment*. Adjacent robots with faulty network connections can detect the faults by combining information from the coordinate system and from local communications and a consistent coordinate system can concatenate x,y,z coordinates into a unique software identifier.

[1] Youtube video link: https://youtu.be/WH_lP3kgpB8

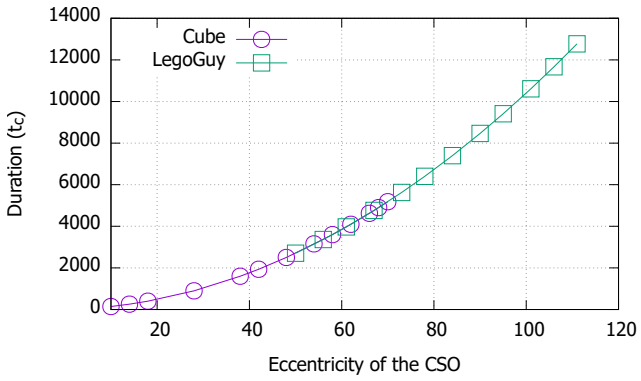

Fig. 7. Duration of the simulation (expressed in function of t_c) depending on the eccentricity of the *CSO* for two shapes: a cube and the Lego Guy model.(Better printed in color)

References

[1] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F. Knight, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, May 2000.

[2] N. Bhalla and C. Jacob. A framework for analyzing and creating self-assembling systems. In *Proceedings of the 2007 IEEE Swarm Intelligence Symposium*, page 281–288, USA, 2007. IEEE Computer Society.

[3] A. Bravais. Mémoire sur les systèmes formés par les points distribués régulièrement sur un plan ou dans l'espace. *Journal de l'École Polytechnique*, 19:1–128, 1850.

[4] Gregory Chirikjian, Amit Pamecha, and Imme Ebert-Uphoff. Evaluating efficiency of self-reconfiguration in a class of modular robots. *Journal of robotic systems*, 13(5):317–338, 1996.

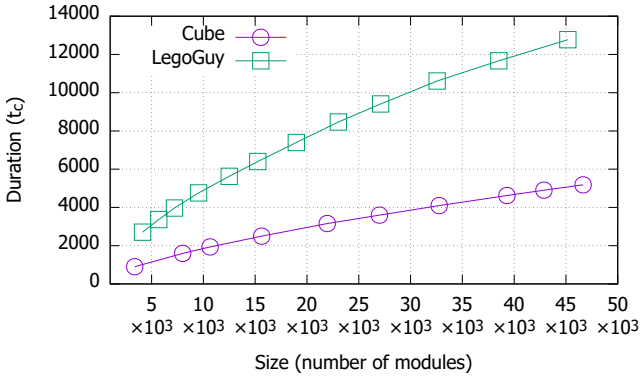

Fig. 8. Duration of the simulation (expressed in t_c) in function of the number of modules for two shapes: a cube and the Lego Guy model.

[5] Daniel Coore. Establishing a coordinate system on an amorphous computer. In *1998 MIT Student Workshop on High Performance Computing in Science and Engineering*, MIT Laboratory for Computer Science, 1998.

[6] Stanislav Funiak, Padmanabhan Pillai, Michael P. Ashley-Rollman, Jason D. Campbell, and Seth Copen Goldstein. Distributed localization of modular robot ensembles. *The International Journal of Robotics Research*, 28(8):946–961, 2009.

[7] Kyle Gilpin, Ara Knaian, and Daniela Rus. Robot pebbles: One centimeter modules for programmable matter through self-disassembly. In *2010 IEEE International Conference on Robotics and Automation*, pages 2485–2492, 2010.

[8] Kyle Gilpin, Keith Kotay, Daniela Rus, and Iuliu Vasilescu. Miche: Modular shape formation by self-disassembly. *The International Journal of Robotics Research*, 27(3-4):345–372, 2008.

[9] Paweł Hołobut, Paweł Chodkiewicz, Anna Macios, and Jakub Lengiewicz. Internal localization algorithm based on relative positions for cubic-lattice modular-robotic ensembles. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3056–3062, 2016.

[10] S. Julier and J. Uhlmann. A non-divergent estimation algorithm in the presence of unknown correlations. *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*, 4:2369–2373 vol.4, 1997.

[11] Brian T. Kirby, Michael Ashley-Rollman, and Seth Copen Goldstein. Blinky blocks: A physical ensemble programming platform. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 1111–1116, New York, NY, USA, 2011. ACM.

[12] John Klingner, Nisar Ahmed, and Nikolaus Correll. Fault-tolerant covariance intersection for localizing robot swarms. *Robotics and Autonomous Systems*, 122:103306, 2019.

[13] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[14] James D. McLurkin. Algorithms for distributed sensor networks. Master's thesis, University of California at Berkeley, 1999.

[15] Radhika Nagpal. Organizing a global coordinate system from local information on an amorphous computer. Technical report, MIT, AI Memo 1666, August 1999.

[16] Radhika Nagpal, Howard Shrobe, and Jonathan Bachrach. Organizing a global coordinate system from local information on an ad hoc sensor network. In *Information processing in sensor networks*, pages 333–348. Springer, 2003.

[17] Andre Naz, Benoit Piranda, Seth Copen Goldstein, and Julien Bourgeois. Abc-center: Approximate-center election in modular robots. In *IEEE RSJ International Conference on Intelligent Robots and Systems (IROS 2015)*, pages 2951 – 2957, Hamburg, Germany, sep 2015.

[18] Andre Naz, Benoit Piranda, Thadeu Knychala Tucci, Seth Copen Goldstein, and Julien Bourgeois. Network characterization of lattice-based modular robots with neighbor-to-neighbor. In *13th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, volume 6 of *Springer Proceedings in Advanced Robotics (SPAR)*, pages 415 – 429, London, United Kingdom, nov 2016. Springer.

[19] Benoit Piranda and Julien Bourgeois. Designing a quasi-spherical module for a huge modular robot to create programmable matter. *Autonomous Robot Journal, Special Issue: 'Distributed Robotics: From Fundamentals to Applications'*, 42(8):1619–1633, 2018.

[20] Benoit Piranda and Julien Bourgeois. Datom: A deformable modular robot for building self-reconfigurable programmable matter. In *15th International Symposium on Distributed Autonomous Robotic Systems (DARS 2021)*, Kyoto, Japan, jun 2021.

[21] Benoit Piranda, Guillaume Laurent, Julien Bourgeois, Cédric Clevy, Sebastian Möbes, and Nadine Le Fort-Piat. A new concept of planar self-reconfigurable modular robot for conveying microparts. *Mechatronics*, 23(7):906 – 915, oct 2013.

[22] Greg Reshko. Localization techniques for synthetic reality. Master's thesis, Carnegie Mellon University, 2004.

[23] John W. Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus. 3d m-blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1925–1932, 2015.

[24] Daniela Rus and Marsette Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[25] A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi, and A.J. Ijspeert. Roombots: A hardware perspective on 3d self-reconfiguration and locomotion with a homogeneous modular robot. *Robotics and Autonomous Systems*, 62(7):1016–1033, 2014. Reconfigurable Modular Robotics.

[26] Pierre Thalamy, Benoit Piranda, Andre Naz, and Julien Bourgeois. Behavioral simulations of lattice modular robots with visiblesim. In *15th International Symposium on Distributed Autonomous Robotic Systems (DARS 2021)*, Kyoto, Japan, jun 2021.

[27] Thadeu Knychala Tucci, Benoit Piranda, and Julien Bourgeois. Efficient scene encoding for programmable matter self-reconfiguration algorithms. In *32nd Annual Symposium on Applied Computing (SAC 2017)*, volume Morroco, Marrakesh of *ACM International Conference Proceedings*, pages 256 – 261, Marrakech, Morocco, apr 2017.