RePoSt: Distributed Self-Reconfiguration Algorithm for Modular Robots Based on Porous Structure

Jad Bassil, Benoit Piranda, Abdallah Makhoul, Julien Bourgeois

RePoSt: Distributed Self-Reconfiguration Algorithm for Modular Robots Based on Porous Structure

Jad Bassil[†], Benoît Piranda[†], Abdallah Makhoul[†], and Julien Bourgeois[†]

***Abstract*— In this paper, we propose a new self-reconfiguration scheme for modular robots based on a meta-module design that allows to form a 3D porous structure. The porous structure enables a parallel flow of modules inside it without blocking. The meta-module can also be used to fill its internal volume with an additional number of modules allowing the structure to be compressible and expandable. Hence, it is a potential for improving the self-reconfiguration process. We first present the meta-module model and the porous structure built using it. Then, we describe an algorithm to self-reconfigure the structure from an initial shape to a given goal shape.**

We evaluated the algorithm in simulation on structures composed of up to 2,700 modules. We studied the performance in term of parallelism, showed that the number of communications is proportional to the number of motions and the execution time varies linearly with the diameter of the configuration.

I. Introduction

A programmable matter can change its physical properties such as its color or even its shape by self-reconfiguring. To achieve such matter, we propose to build it using an ensemble of autonomous and communicating tiny robots which brings the capability of distributed computation to the physical object itself. Furthermore, unlike the atoms of the real matter which are strongly linked to each other, the modules that form the programmable matter have displacement capabilities which allow, by local movements, the matter to be reshaped. For instance, Roombots [1] is a modular robotic system that is formed by modules that can autonomously connect to each other to build furniture-like structures capable of autonomously moving, self-reconfiguring and self-repairing. However, having mobile connected robots is not enough to obtain a self-reconfigurable programmable matter. If we consider that the robots initially build a shape A and must at the end self-reorganize to build a shape B, a subset of these robots has to move while avoiding collisions and maintaining the connectivity of the assembly.

Self-reconfiguration algorithms [2] aims to transform the initial shape of a modular robot into a given goal shape using motions coordinated by communications. Self-reconfiguration is an intricate task since the size of configuration space increases exponentially with the size of the ensemble. It is an operation which can take a lot of time because it involves a lot of movement of modules, as for example, in the situation of Kilobots [3], it could take from 6 to 12 hours to reconfigure 1,000 Kilobots.

[†]All authors are with Univ. Franche-Comté, FEMTO-ST Institute, CNRS, 1 cours Leprince-Ringuet, 25200, Montbéliard, France. {first}.{last}@femto-st.fr

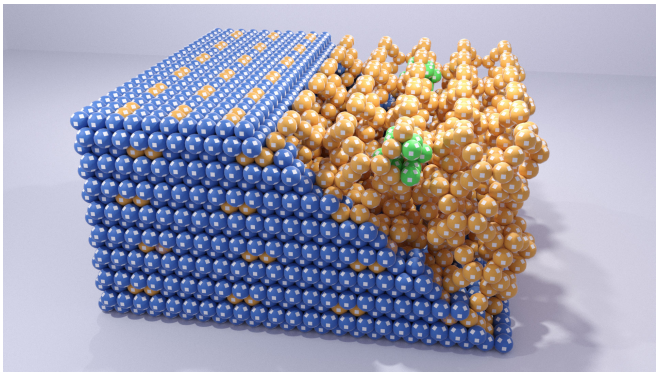


Fig. 1: A box made of *3D Catoms*, we remove top right border modules to better present the internal structure. The porous structure is made of meta-modules that construct a scaffold in orange, some reserves of modules are drawn in green and border meta-modules are drawn in blue.

Two main optimizations are to reduce the distance traveled by the modules to reach their goal position and to allow simultaneous motions of as many modules as possible. Lengiewicz et al. [4] proposed an algorithm that adapts the maximum flow search algorithm to define independent paths along which robots can move in parallel in porous structures. Another solution has been proposed by Thalamy et al. [5] with a scaffolding structure that predefines independent paths allowing the parallelization of moves.

An application of such a self-reconfigurable programmable matter is an electronic clay that makes a digital twin synchronized with real matter. Electronic clay can be deformed manually or by command to create a tangible digital 3D model of an object. The solution proposed in this paper may allow a quick update of the object by providing a reserve of matter close to the borders and increasing the simultaneous movements of modules.

In this work we present a self-reconfiguration scheme based on a porous structure formed using meta-modules grouping several quasi-spherical robots able to move by rolling on their neighbors: the *3D Catoms* [6]. This original meta-module shape was briefly introduced in our work in [7]. Internal meta-modules construct a scaffolding structure drawn in orange in Figure 1 that helps with the displacement of modules inside the structure and at the same time allows to store modules in the structure. The structure will be coated by solid meta-modules (drawn in blue) in order to close the volume.

Using these meta-modules we obtain a cubic lattice at

the meta-module scale composed of three-state cells that can be "EMPTY", "SPARSE" or "FULL". "SPARSE" meta-modules allow modules to flow in parallel through their internal volume without blocking. A "FULL" meta-module provides a new functionality: it is a storage place or a reserve of material. Filling a meta-module allows to reduce the distances covered during self-reconfiguration, either by proposing outgoing modules closer to their delivery places or by providing a place to store incoming modules. Our self-reconfiguration solution exploits the properties of this structure to both reduce the distances covered by the robots and to parallelize their movements.

In the following sections, we will first detail our meta-module model by presenting the basic operations to perform to change its state including the dismantling, transfer and reassembly of the meta-module (Section IV). And in the second part, we will propose *RePoSt*: a self-reconfiguration algorithm based on the "maximum flow search" algorithm to determine a set of paths allowing simultaneous movement of our meta-modules. It is based on a system of agents that can have 4 different states during the reconfiguration process (Section V).

II. RELATED WORKS

Many tunneling-based reconfiguration algorithms that use meta-modules have been proposed in the literature [8, 9, 10, 11]. Tunneling allows in-place reconfiguration where modules flow in parallel through the internal volume of the ensemble. They are specific to deformable modules that can contract and expand, which requires a specific hardware design. In most of them, modules are grouped into meta-modules seen as a unit to facilitate both planning and motion operations. Parada et. al [12] described a meta-module design that simulates the contract/expand capability. It can be applied to a wide range of non deformable modular robots to enable tunneling. Nonetheless, its only suitable for modules placed in square cubic lattice.

Dewey et. al [13] described a generalized model for meta-modules independent from module's design which inspired this work. It aims to achieve a holonomic system where modules are arranged in meta-module units that can be in two states: filled or empty. Modules flow from a filled meta-module to an empty one to reach their goal position guided by a planner.

Self-reconfiguration algorithms have been proposed in which modules are arranged in a regular porous structure leaving enough internal space for modules to flow through it to facilitate motion and coordination have been proposed in [14, 15, 16, 17]. However, they use structures applicable only in a square cubic lattice.

Lengiewicz et. al [17] solved reconfiguration planning through max-flow search to find the optimal number of disjoint streamlines between boundaries for modules to flow in concurrently. In this work, we use their same planning approach, but we are not restricted to boundaries. Our meta-module design allows modules to start flowing from any position in the configuration.

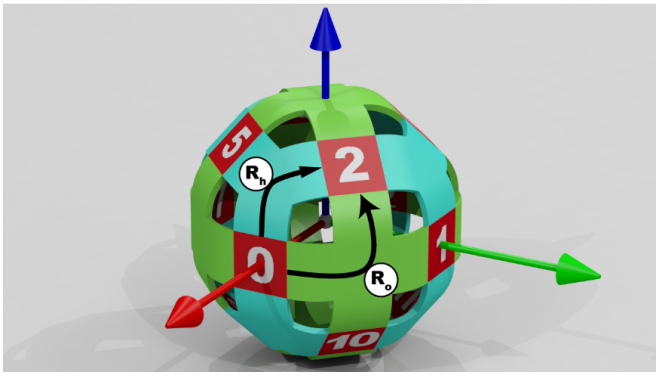


Fig. 2: A *3D Catom* with two rotation methods [6]. The first on the octagonal surface area R_o and the second on the hexagonal surface area R_h.

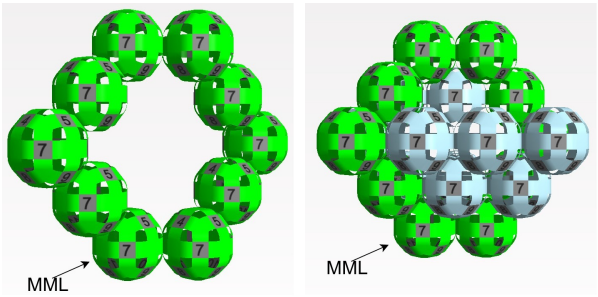
The most relevant self-reconfiguration work to ours is described in [5], in this paper the authors propose a self-reconfiguration scheme for modular robots based programmable matter. Modules flow from a reserve of modules called *Sandbox* to build the internal scaffold of an object. The object is then coated with a thin layer of modules to present high fidelity to the goal shape. To the best of our knowledge, it is the only work on self-reconfiguration that uses the same modular robotic system as ours: *3D Catoms* [6] placed in a face-centered cubic lattice. However, it only describes how we can initially build an object from a reserve of modules. Our interest in this paper is in reconfiguring the structure from its initial shape to a given goal shape.

In our work, we are focusing on proposing a distributed self-reconfiguration scheme based on a 3D porous structure constructed of *3D Catoms* placed in a face-centered cubic lattice based on a new meta-module design. It allows tunneling and offers the ability to store an additional number of modules in its empty volume which can be exploited in multiple ways to enhance the self-reconfiguration process in the endeavour to achieve a programmable matter.

III. MODULAR ROBOTIC MODEL

In this work, we consider the self-reconfiguration of modular robots made of *3D Catoms*. *3D Catoms* are quasi-spherical micro-robots introduced by Piranda et. al [6]. They can be arranged in a 3D regular grid described as a face-centered cubic lattice (FCC). Each *3D Catom* has 12 connectors used to latch to and communicate with up to 12 neighbors. A *3D Catom* can move from a position to an adjacent one by rotating on a fixed neighbor module acting as a pivot using the electrostatic actuators situated at the orthogonal and hexagonal surfaces as shown in Figure 2. Prototypes of *3D Catom* are currently not available in number but they may be simulated by the *VisibleSim* simulator [18].

A *3D Catom* motion is subject to the following local constraints which need a motion coordination mechanism to overcome:



(a) "SPARSE" (b) "FULL"

Fig. 3: Meta-module's anatomy.

- Collision constraint: A moving *3D Catom* should not collide with another moving *3D Catom*.
- Blocking constraint: The motion of a *3D Catom* entering a free grid cell should not be blocked by any other module belonging to its neighborhood or not.
- Bridging constraint: A module cannot enter a free target cell in the grid if there are two modules in opposite directions adjacent to the target cell.

The connected ensemble of *3D Catoms* forms a distributed system where:

- All *3D Catoms* execute the same distributed program and perform the computations locally to each module.
- All communications are done in a local fashion where a module can only communicate with its direct connected neighbors when it is docked by exchanging messages.
- The interconnections graph must be connected all the time. This adds an additional constraint to be considered by a self-reconfiguration algorithm.
- All modules share the same coordinate system. Each module stores its coordinates locally and updates them after each movement.
- A *3D Catom* can react to the reception of a message, the connection and disconnection of a neighbor and any internal event such as a timer event or a rotation end event.

IV. META-MODULES ANATOMY

The meta-modules are composed of 10 attached *3D Catoms* forming a 3D hexagon shape. The meta-module can be in two states "SPARSE" or "FULL" as in Figure 3. A "FULL" meta-module groups 20 *3D Catoms* by storing in its empty internal volume 10 additional modules. We have experimentally verified that the size 10 is the smallest size that allows the free flow of modules in the internal volume of a "SPARSE" meta-module without being blocked by the modules of the "SPARSE" meta-module that they are traversing through. The meta-modules can be attached to form a 3D structure in a 3D cubic lattice as shown in Figure 4.

Filling a meta-module allows the structure to be compressible or expandable by a factor of 2 since each "SPARSE" meta-module can store in its empty volume the size of another meta-module and the "FULL" meta-module can expand by discarding its filling modules so they can be reassembled at a free position.

This structure enables modules to flow freely inside of it. In addition, all meta-modules, "SPARSE" or "FULL", have at least one module who is not blocked, free to move, therefore it can be dismantled and transported in any direction regardless of its position. Furthermore, the compressibility and the expandability property of the structure does not constrain the goal shape to have exactly the same number of meta-modules as the initial shape.

Let N be the total number of modules in the initial shape and S_G the size of the goal shape in terms of meta-module ("FULL" or "SPARSE"). Due to the expandability and compressibility properties of the proposed structure: $\lceil \frac{N}{20} \rceil \leq S_G \leq \frac{N}{10}$.

To change the shape of the meta-modules structure, we use three basic operations that can be executed in the six directions (up, down, front, back, left and right) in the meta-modules scale cubic lattice by a meta-module to change the state of its cell:

1) *Dismantle* operation changes the state of a cell from "SPARSE" to "EMPTY" or "FULL" to "SPARSE" by breaking or emptying the meta-module and transporting its modules to an adjacent cell.
2) *Transfer* operation does not change the state of a cell. It is only used to transport the modules through a "SPARSE" cell.
3) *Assemble* operation changes the state of a cell from "EMPTY" to "SPARSE" or "SPARSE" to "FULL".

Each movement is coded by a triplet in the form of <*current_position, next_position, state*> where the three possible values of *state* are:

1) MOVING: to indicate that the module must keep moving when *next_position* is reached.
2) WAITING: to indicate that the module must stop and wait when *next_position* is reached to serve as a bridge for next flowing modules, or to wait when filling a meta-module before entering its final position to avoid blocking.
3) IN_POSITION: to indicate that the module will reach the final position for current operation.

All the moving operations can be applied by a *3D Catom* in playing a sequence of basic movements. All the sequences must be pre-stored in the robot memory for the six possible directions of motion. However, some operations can be deduced from others. For example, a module needs only to store the movements of the *Transfer* operations in 3 directions e.g. up, left and back. The movements for the other directions can be deduced from the stored ones by executing them in reverse order. In addition, the *Assemble* and *Dismantle* operations are homologous, the movements required to dismantle a meta-module are the same as the ones required to build it but in reversed order. Hence, reducing the number of operations to be stored in each module. Finally, each module must store 2.24 kB of predefined movement data. The details of the coding of the triplet in memory is

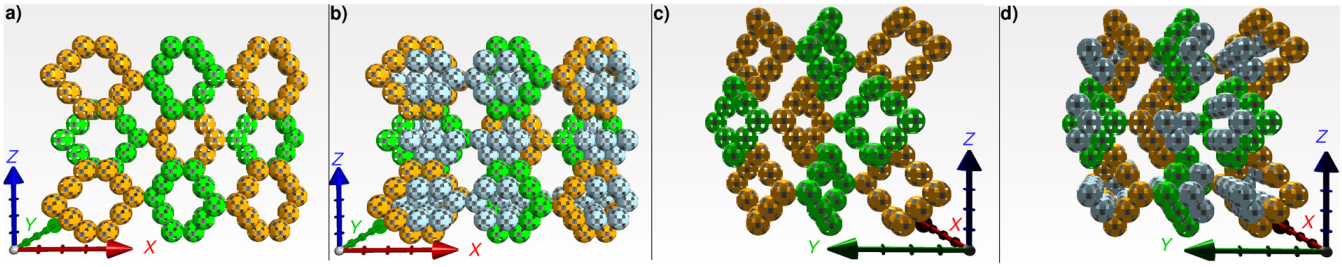


Fig. 4: Meta-modules structure in a 3D cubic lattice. a) "SPARSE" meta-modules in the *XZ* plane. b) "FULL" meta-modules in the *XZ* plane. c) "SPARSE" meta-modules in the *YZ*. d) "FULL" meta-modules in the *YZ* plane.

proposed in Section V-D.1.

These operations can be exploited by a self-reconfiguration planner who's purpose will be to specify what operation to execute on which meta-module.

The video[1] presents in its first part the several possible operations. First, it shows the *Dismantle* operation on a "SPARSE" meta-module whose modules are reassembled at a "SPARSE" cell that becomes "FULL". Second, the "FULL" meta-module discard its filling modules to be transferred then reassembled at an "EMPTY" cell that becomes "SPARSE".

V. SELF-RECONFIGURATION SCHEME

In this section, we describe *RePoSt*: a self-reconfiguration algorithm for transforming a structure composed of the meta-modules described in Section IV from its current shape to a given goal shape.

The *RePoSt* algorithm is ran in all modules of the system. This algorithm is mainly divided into three steps repeated sequentially until the convergence to the goal shape:

1) Determining sources and destinations meta-modules.
2) Finding the maximum number of possible streamlines connecting sources and destinations.
3) Dismantling sources meta-modules and transporting their composing modules to destinations.

The steps are detailed in the next subsections.

During the reconfiguration process an agent can take four different roles. For each of these roles it runs a corresponding code:

[1] YouTube video: https://youtu.be/6U3Wsroy-f8

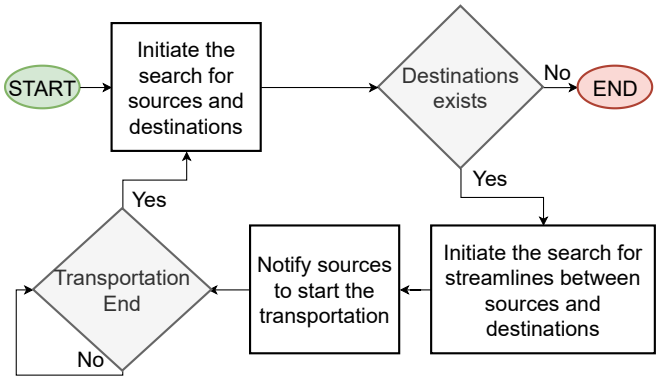


Fig. 5: *Global Coordinator* agent's flow chart

1) *Global Coordinator* (GC) module coordinates the sub-stepping scheme. It is a fixed module that belongs to a meta-module initially in the goal shape so it will not change its position during the reconfiguration process. At each iteration the *Global Coordinator* initiates each step, detects its termination then initiates the next one until the goal shape is achieved as shown in Figure 5. The termination of the reconfiguration process is determined if no destinations are found after the first step.
2) *Meta-Module Leader* (MML) is a module chosen in each meta-module which handles computations and communications between meta-modules. Messages between meta-modules are routed from a MML to another. The MML can be any module in the meta-module and in our scenario we choose the bottom left one as presented in Figure 3.
3) *Operation Coordinator* (OPC) is the module that coordinates the sequence of movements executed on its meta-module along the streamline as described in Section V-C.
4) *Flowing Module* (FM) is any module in motion flowing along a streamline to reach its destination.

We assume that each module knows in addition to its coordinates in the face-centered cubic lattice, the coordinates of its meta-module in the square cubic lattice. This can be efficiently disseminated by the GC whose coordinates are the origin. On reception a MML set the coordinates of its meta-module according to the direction of the sender. Moreover, all meta-modules know the goal shape and can determine if they are in it or not. This can be efficiently done using the method described in [19] applied at the meta-module scale where a module is replaced by a meta-module and the computations are done by the MMLs.

A. Determination of sources and destinations

To reconfigure the system from its current shape to a given goal one, we must first determine sources and destinations. Sources are any meta-modules that are "FULL" or do not belong to the goal shape. A "SPARSE" source can be dismantled and a "FULL" source can discard its filling modules. The modules of a source are transported to a destination meta-module. Destinations are "SPARSE" meta-modules adjacent to an empty cell belonging to the goal shape that needs to

be occupied or a "SPARSE" meta-module to be filled. When a source's module arrives to a destination, the destination handles its transportation to its meta-module goal position. We assume that each meta-module must know the goal shape and can determine locally if it belongs to it or not.

1) Sources Determination: A source is defined as a "SPARSE" meta-module in the initial shape that does not belong to the goal shape or a "FULL" meta-module at any position. Initially, all "SPARSE" meta-modules that do not belong to the goal shape are potential sources. Then, a potential source is confirmed to be a source if it does not disconnect the structure after being dismantled. We use the connectivity preservation method described in [17] to choose the meta-modules that when dismantled do not disconnect the structure. Briefly, the connectivity preservation method consists of building a tree rooted at the GC in a way that the leaves meta-modules, when removed, do not disconnect the structure. So, the tree leaves, if they are potential sources, are confirmed to be sources.

2) Destinations determination: Potential destinations are meta-modules adjacent to empty positions in the goal shape or "SPARSE" meta-modules to be filled if the initial shape is larger than the goal shape. A source meta-module will be dismantled and transported to a destination meta-module that coordinates the reassembling process of the source.

A problem that can occur is having an empty goal position adjacent to multiple potential meta-modules destinations as seen in Figure 6. When a potential destination is determined, to avoid collision it must be a destination for only one empty goal position. One solution is to report back to the GC which empty goal positions a destination corresponds. In its turn, the GC chooses one empty goal position for each potential destination and notifies each destination about its associated empty goal position.

B. Finding streamlines

After the sources and destinations are determined, the maximum number of streamlines connecting sources and destinations is found. A streamline is defined as a path of adjacent meta-modules that starts from a source and ends at a destination. Streamlines must be disjoint to avoid collisions at intersections. This can be achieved by solving the classical problem of maximum-flow in graphs between many sources to many destinations with a unit edge capacity. A distributed asynchronous version of the Edmond-karp max-flow algorithm [20] proposed in [17] is used and adapted for this purpose. Each source initiates a breadth-first search for destinations. When a destination is reached a unique path is backtracked and the unused branches of the tree are cut off leaving a place for other trees to grow. Then, the source confirms the path by sending a message along it to confirm the streamline. The algorithm terminates when no additional streamlines can be found. The GC needs to detect the termination of this step before proceeding to modules transportation. We use the distributed termination detection method described in [21].

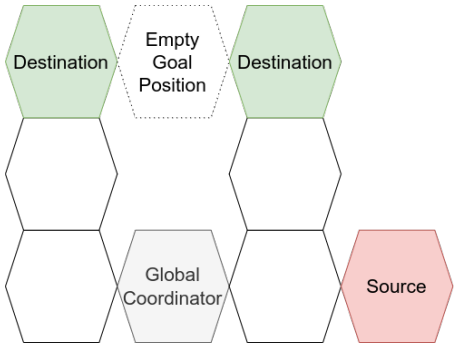


Fig. 6: Two potential destinations for one empty goal position.

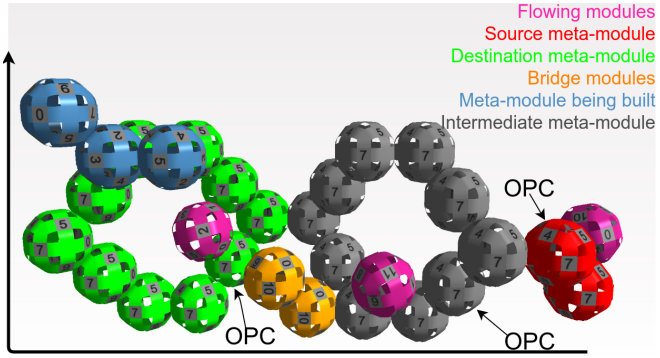


Fig. 7: Simulation snapshot during modules transportation (best viewed in color). The source at the right is dismantled to be built back at the top of the destination on the left.

C. Modules transportation

After the establishment of disjoint streamlines connecting sources and destinations, sources must be dismantled and transported along the streamlines to the destinations. This is done using the meta-modules operations described in Section IV.

Each MML in a streamline knows the position of the previous and next meta-module in the same streamline. This information is used to determine the direction of the operation to be executed on each meta-module. The *Dismantle* operation is executed on source meta-modules, *Transfer* operation is executed on intermediate meta-modules to transfer modules to the next meta-module in streamline and *Assemble* operation is executed at destinations. Figure 7 shows modules transportation from a source on the right to a destination on the left. In this figure, the modules in orange are in a waiting state serving as a bridge for purple moving modules to flow without blocking.

To avoid blocking and collisions during the flow of multiple modules, modules flow in one line following the same path. A message-passing traffic-light style motion coordination protocol described in [22] is used to keep a space gap between every two moving modules. Moreover, the structure described in Section IV keeps enough space between meta-modules to allow the modules to flow in parallel in multiple

adjacent streamlines without collisions.

Each moving module performing an operation keeps an iterator on the sequence of movements of the operation being executed. A MML in the streamline assigns a module in its meta-module as an OPC according to the operation to be executed. For the *Dismantle* operation, it is the last module connected to the meta-module in the operation's direction. For all the other operations, it is the first module that a moving module from previous operation get connected to. The algorithm executed by the FMs is described in Algorithm 1. When a FM module is attached to the OPC, it means that it ended the previous operation and is ready to start executing the sequence of movements of the next operation. So, the OPC sends the COORDINATE_MSG containing the operation to be executed by the FM and the value of the iterator so the FM knows from which movement it must begin. On reception, the FM will start moving until reaching the state IN_POSITION meaning that it ended the movements to be executed for the current operation or WAITING meaning that the module must stop and serve as a bridge for next modules to pass it (Algorithm 1 line 6-14). If the operation is a *Dismantle* operation, when a FM becomes IN_POSITION, it notifies its operation coordinator so it can proceed to the next module. Note that the function *rotateTo* executed by a FM encompasses the motion coordination algorithm that requires additional exchanged messages between the moving module, its pivot and its future latching points. The reader can refer to [22] for detailed description of the motion coordination algorithm.

Algorithm 1: Distributed control algorithm for a *Flowing Module*

```
   Data: Operation: The operation in execution.
   Data: mvt_it = 0: Iterator on Operation's movements.
1  Msg Handler COORDINATE_MSG(Op, it):
2      Operation ← Op ;
3      mvt_it ← it ;
4      rotateTo(Operation[mvt_it].nextPosition) ;
5  Event ROTATION_END:
6      if mvt_it = Operation.size ∧ Operation.isAssemble
         then
7          meta-module reached goal position;
8      else
9          if Operation.state = MOVING then
10             mvt_it ← mvt_it + 1 ;
11             rotateTo(Operation[mvt_it].nextPosition) ;
12         else
13             if Operation.isDismantle ∧ Operation.state =
                 IN_POSITION then
14                 Notify the Operation Coordinator;
15 Event REMOVE_NEIGHBOR:
16     if Operation.state = WAITING then
           // Bridge
17         if all modules have passed then
18             mvt_it ← mvt_it + 1 ;
19             rotateTo(Operation[mvt_it].nextPosition);
```

D. Complexity

In this section, we detail the memory needed and the number of basic treatments used by the algorithm in order to express the complexity of our method in terms of memory and time.

1) Memory complexity: Meta-modules execute basic operations to transform the shape of the structure. Therefore, each module must store in its memory a database containing the sequence of movements to be executed for each operation. Table I shows the number of movements for the *Assemble* operations into a "SPARSE" or "FULL" meta-module and *Transfer* operations that must be stored in a module. Other operations can be deduced from the stored one as mentioned in Section IV.

Each movement is stored in the database using 4 bytes in order to embed the $6 \times 4bits$ used for coordinates plus $2bits$ for the state. As presented in the Table I, there are 559 records in total stored in the movements database requiring 2.24 kB.

It is important to note that despite the fact that this memory is quite large in the context of modular robots, it is a constant size. The other variables used locally for the self-reconfiguration algorithm are also constant. This allows us to obtain a constant complexity in memory for this algorithm.

2) Time complexity: As presented in section V, the algorithm repeats M rounds, which consists in transporting modules along the streamlines. The value of M varies enormously in function of the initial, intermediate and final shapes of the self-reconfiguration. It is mainly affected by the number of streamlines that can be established at each iteration.

We consider that the duration of a basic motion can be majored by the time t_m and that the duration of the whole self-reconfiguration is mainly due to the number of motions which are much longer than the communications times.

The longest stage is the motions of modules along the longest streamline of a round. The length of such a streamline can be majored by the diameter of the configuration divided by the diameter of a meta-module, we call d this dimension. Considering that the number of movements applied for *Dismantle* and *Assemble* operations can be majored by N_0 and that the number of movements applied for a *Transfer* operation can be majored by N_1, we can express the number

TABLE I: Number of movements per operation

Operation	Direction	Nb of Movements
Assemble ("SPARSE")	Up/Down	41
Assemble ("SPARSE")	Right/Left	59
Assemble ("SPARSE")	Back/Front	64
Assemble ("FULL")	Up/Down	72
Assemble ("FULL")	Right/Left	78
Assemble ("FULL")	Back/Front	91
Transfer	Up/Down	50
Transfer	Right/Left	60
Transfer	Back/Front	44
		Total: 559

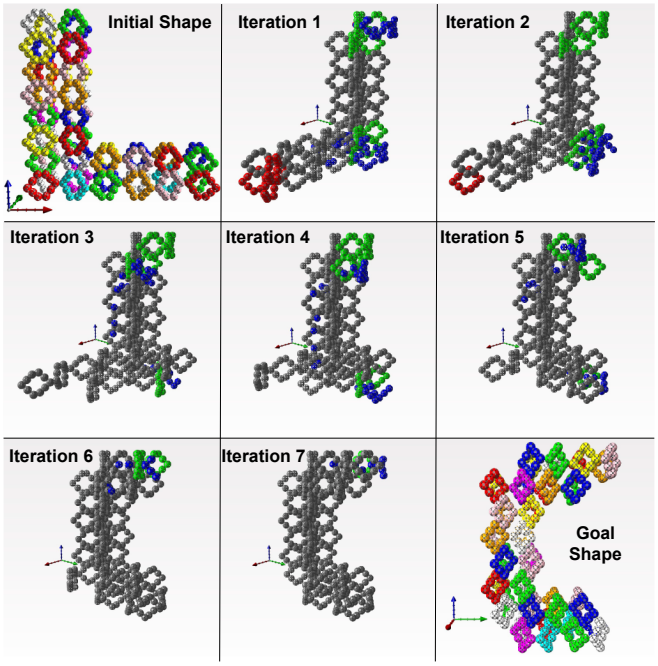


Fig. 8: Simulation snapshots for the 7 iterations during the reconfiguration of 48 meta-modules in an L shape to a C shape.

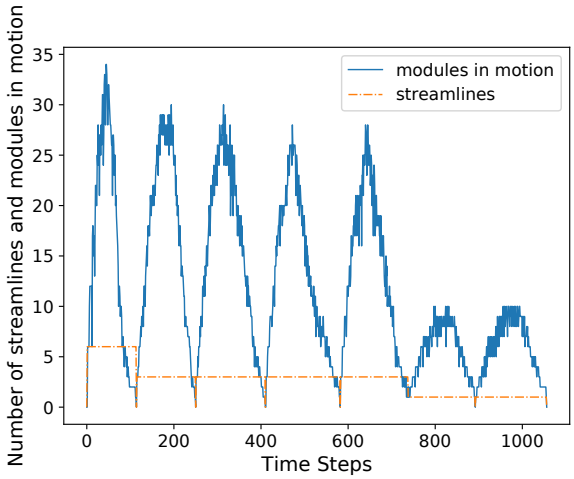


Fig. 9: Motion parallelism and number of streamlines during the reconfiguration of 48 meta-modules from a L shape to a C shape.

of basic motions realized to cross a streamline of length x by: $N_{motions} = 2 \times N_0 + (x-1) \times N_1$.

The longest streamline being d long, we can majorize x by d, then we have: $N_{motions} < 2 \times N_0 + (d-1) \times N_1$. Therefore, the time complexity can be expressed as $O(M \times N_{motions} \times t_m)$.

VI. Experiments

In this section, we aim to show the functioning of our method and show the different operations realized in parallel by the meta-modules along the streamlines. Moreover, we show that our algorithm is capable of using the predefined operations to reconfigure a structure from an initial shape to a goal one in a 3D space.

All simulations are done in *VisibleSim* [18]: a discrete-event simulator for modular robots. Figure 8 shows snapshots of simulation during the reconfiguration of an L shape made of 48 meta-modules placed in the XZ plane to a C shape in the YZ plane.

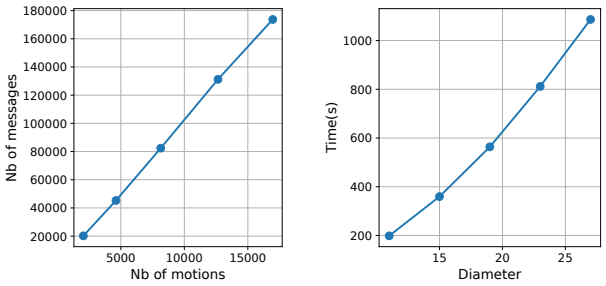
The video[1] shows in its second part a simulation of the reconfiguration of 270 "Sparse" meta-modules placed in a 3 layer square shape into a humanoid shape of size 267 meta-modules. The additional 3 meta-modules in the initial shape are filled inside the structure during the last iterations. In its last part, the video shows the expansion of a $6 \times 6 \times 3$ configuration with "Full" meta-modules at the bottom layer into a $6 \times 6 \times 4$ configuration. All "Full" meta-modules are emptied and their filling modules are transported in parallel to form an additional layer of "Sparse" meta-modules at the top of the configuration in one iteration.

Figure 9 shows the number of streamlines and the number of modules moving concurrently along the streamlines against time steps during the reconfiguration example of Figure 8. One time step corresponds to the time taken by a module to move from one position in the grid to an adjacent one. Each bell curve in the graph corresponds to an iteration. The number of streamlines and motions becomes null between two peaks which corresponds to the time required for the first two steps of the algorithm: finding sources and destinations and determining the streamlines. It can be seen that it is negligible compared to the time required to transport the modules. The maximum number of concurrent motions corresponds to the size of the meta-module times the number of streamlines meaning that all the modules of the dismantled source are moving at the same time. It reaches the maximum from iteration 2 to 7. At iteration 1 it is less than the maximum because some modules have reached their goal position while others have not yet started their movements.

Figure 10 evaluates the number of communications and the time for the reconfiguration of an L shape to a C shape while varying the size of the configuration. Figure 10a shows that the number of exchanged messages is proportional to the number of motions which increases when the configuration size increases. Figure 10b shows that the execution time increases linearly in the diameter of the system.

VII. Conclusion and Future Work

In this paper, we proposed *RePoSt*: a new self-reconfiguration scheme for *3D Catoms* based on a 3D porous structure composed of 3 states meta-modules. We first presented the meta-modules anatomy and showed how it can be exploited to expand or compress the structure by filling it with a reserve of modules and improve the self-reconfiguration process in terms of parallelism of motions. We then described a distributed multi-agent self-



(a) Exchanged Messages (b) Simulation time

Fig. 10: Number of exchanged messages vs number of motions and simulation time versus the diameter of the system for the reconfiguration of an L shape to a C shape while varying the configuration sizes in $\{20, 28, 36, 44, 52\}$ meta-modules

reconfiguration algorithm to change the shape of the structure into a goal one by dismantling meta-modules, transferring their composing modules along disjoint streamlines and rebuilding them in empty positions in the goal shape. We showed examples of reconfiguration in simulation and studied the motion parallelism. We also provided a communication and time analysis and showed that the number of exchanged communications is linear in the number of motions and the time is linear in the diameter of the ensemble.

In future works, we aim to study the expandability and compressibility properties of our structure and how they can be exploited to enhance the self-reconfiguration process. For instance, during an iteration, empty meta-modules near the goal shape can be filled, then its filling modules are transported during the next iteration so the distance from source meta-modules to the goal shape is reduced. In addition, we aim to provide a solution to better represent the goal object. Perhaps, we can change the shape of boundary meta-modules to better mimic the boundary of the goal shape and use filling modules for coating it.

ACKNOWLEDGMENT

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

REFERENCES

[1] A. Spröwitz, P. Laprade, S. Bonardi, M. Mayer, R. Moeckel, P. A. Mudry, and A. J. Ijspeert, "Roombots—Towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Oct. 2010.

[2] 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. [Online]. Available: https://doi.org/10.1016/j.robot.2019.07.012

[3] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3293–3298.

[4] J. Lengiewicz and P. Hołobut, "Efficient collective shape shifting and locomotion of massively-modular robotic structures," *Autonomous Robots*, Feb. 2018. [Online]. Available: https://doi.org/10.1007/s10514-018-9709-6

[5] P. Thalamy, B. Piranda, and J. Bourgeois, "Engineering efficient and massively parallel 3d self-reconfiguration using sandboxing, scaffolding and coating," *Robotics and Autonomous Systems*, vol. 146, p. 103875, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889021001603

[6] B. Piranda and J. Bourgeois, "Designing a quasi-spherical module for a huge modular robot to create programmable matter," *Autonomous Robots*, vol. 42, no. 8, pp. 1619–1633, dec 2018. [Online]. Available: http://link.springer.com/10.1007/s10514-018-9710-0

[7] J. Bassil, B. Piranda, A. Makhoul, and J. Bourgeois, "A new porous structure for modular robots," in *Proceedings of the 21th International Conference on Autonomous Agents and MultiAgent Systems, 2022. (Extended abstract)*.

[8] S. Vassilvitskii, M. Yim, and J. Suh, "A complete, local and parallel reconfiguration algorithm for cube style modular robots," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 1, 2002, pp. 117–122 vol.1.

[9] H. Kawano, "Tunneling-based self-reconfiguration of heterogeneous sliding cube-shaped modular robots in environments with obstacles," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 825–832.

[10] ——, "Distributed tunneling reconfiguration of sliding cubic modular robots in severe space requirements," in *Distributed Autonomous Robotic Systems*, N. Correll, M. Schwager, and M. Otte, Eds. Cham: Springer International Publishing, 2019, pp. 1–15.

[11] ——, "Distributed tunneling reconfiguration of cubic modular robots without meta-module's disassembling in severe space requirement," *Robotics and Autonomous Systems*, vol. 124, p. 103369, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889019301447

[12] I. Parada, V. Sacristán, and R. I. Silveira, "A new meta-module design for efficient reconfiguration of modular robots," *Autonomous Robots*, pp. 1–16, mar 2021. [Online]. Available: https://doi.org/10.1007/s10514-021-09977-6

[13] D. J. Dewey, M. P. Ashley-Rollman, M. De Rosa, S. C. Goldstein, T. C. Mowry, S. S. Srinivasa, P. Pillai, and J. Campbell, "Generalizing metamodules to simplify planning in modular robotic systems," *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 1338–1345, 2008.

[14] K. Kotay and D. Rus, "Algorithms for self-reconfiguring molecule motion planning," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, vol. 3, 2000, pp. 2184–2193 vol.3.

[15] C. Unsal and P. K. Khosla, "A multi-layered planner for self-reconfiguration of a uniform group of i-cube modules," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium*, vol. 1. IEEE, 2001, pp. 598–605.

[16] K. Stoy and R. Nagpal, "Self-reconfiguration using directed growth," in *Distributed Autonomous Robotic Systems 6*, R. Alami, R. Chatila, and H. Asama, Eds. Tokyo: Springer Japan, 2007, pp. 3–12.

[17] 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. [Online]. Available: https://doi.org/10.1007/s10514-018-9709-6

[18] P. Thalamy, B. Piranda, A. Naz, and J. Bourgeois, "Behavioral simulations of lattice modular robots with visiblesim," in *15th International Symposium on Distributed Autonomous Robotic Systems (DARS 2021)*, Kyoto, Japan, jun 2021. [Online]. Available: https://publiweb.femto-st.fr/tntnet/entries/17403/documents/author/data

[19] T. Tucci, B. Piranda, and J. Bourgeois, "Efficient scene encoding for programmable matter self-reconfiguration algorithms," *Proceedings of the ACM Symposium on Applied Computing*, vol. Part F1280, pp. 256–261, 2017.

[20] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, p. 248–264, Apr. 1972. [Online]. Available: https://doi.org/10.1145/321694.321699

[21] J. Brzezinski, J. Helary, and M. Raynal, "Termination detection in a very general distributed computing model," in *[1993] Proceedings. The 13th International Conference on Distributed Computing Systems*, 1993, pp. 374–381.

[22] P. Thalamy, B. Piranda, F. Lassabe, and J. Bourgeois, "Deterministic scaffold assembly by self-reconfiguring micro-robotic swarms," *Swarm and Evolutionary Computation*, vol. 58, p. 100722, 2020.