Adaptive Heuristics for Obstacle Handling and Uncertainty in Modular Robots

Benoît Piranda, Julien Bourgeois, Jacques Demerjian and Abdallah Makhoul

Abstract Modular self-reconfigurable (MSR) robots have the ability to change their structure through processing, communication, and movement in a process called self-reconfiguration. Although dynamic self-reconfiguration has shown promising results in different applications, robots still face uncertainties while performing tasks in the real world. For example, they may encounter unexpected obstacles that can divert them from their goals or even cause the system to shut down. This paper introduces a new approach to help robots detect and adapt to obstacles, allowing them to continue achieving their original objectives. This method focuses on the movement and direction of two-dimensional self-reconfigurable robots. We use VisibleSim to test and evaluate this algorithm in different scenarios, from dealing with a single obstacle to navigating a full maze.

Key words: Programmable Matter, Modular Robots, Distributed Algorithms, 2D Self-Reconfiguration, Uncertainty, Obstacles.

1 Introduction

Programmable Matter (PM) refers to materials that are able to alter their physical properties, such as shape, color, or density according to the surrounding environment. Modular robotics refers to robots with interchangeable parts that can perform a variety of functions. Modular self-reconfigurable robots (MSRs) are distributed systems composed of a large number of autonomous modules that work together

Benoit Piranda, Julien Bourgeois, Abdallah Makhoul
Université Marie et Louis Pasteur, CNRS, institut FEMTO-ST, F-25200 Montbéliard, France e-mail: `first.last@femto-st.fr` Jacques Demerjian
LaRRIS, Faculty of Sciences, Lebanese University, Fanar P.O. Box 90656, Lebanon and
Computer Science & IT Department, Faculty of Arts and Sciences, Holy Spirit University of Kaslik (USEK), Jounieh P.O. Box 446, Lebanon

to form a collective structure and achieve a shared goal [1, 2]. Each module is equipped with similar computational, communication, sensory, and actuation capabilities, making them both physically and logically independent [3, 4]. Several architectures of MSR systems are proposed in the literature (lattice, chain/tree, and mobile) [5, 6] and they can deterministic or stochastic depending on their configuration. In this work, we employ a lattice architecture, where modules align on a structured grid, and we consider a deterministic configuration, where modules move in coordinated steps toward a predefined target positions.

One of the primary challenges faced by MSRs is adapting the movements of modules to obstacles while maintaining progress toward their target location [7]. In this paper, we tackle the obstacle detection problem using a self-reconfiguration algorithm allowing a 2D self-reconfigurable robot system to transform itself from an initial shape (shape A) into a goal shape (shape B). This algorithm enables the robots to detect potential obstructions by monitoring their movement direction and evaluating their next prospective positions. Upon identifying an obstacle, the MSR halts and neighboring modules communicate together, allowing the system to adapt and continue without collisions or interruptions. To evaluate our proposal, we implemented and tested it using Hexanodes 2D robots within the VisibleSim simulator [8]. We tested various scenarios with differing obstacle placements, initial and target shapes, and the number of modules. The performance is demonstrated by analyzing the number of messages exchanged and the time taken for modules to reach their target positions in each scenario.

The remainder of this work is organized as follows: We discuss some earlier work in the Section 2. Section 3 introduces the 2D self-reconfigurable robots and its challenges and presents our proposed self reconfiguration algorithm. The simulation and experimental findings are presented in the Section 4. Finally, Section 5 concludes the paper while suggesting some additional research and improvements.

2 State of the Art

MSRs (modular self-reconfigurable robots) have evolved into a major topic in the robotics field. These MSR systems are made up of a huge number of identical robots that can communicate with one another and change the system's topology [9]. Each self-reconfigurable robot can execute a limited set of activities, but when linked together, they can adapt and perform considerably more difficult and complex jobs [7]. As a result, these systems can alter their structure dynamically in order to complete the assigned work. This is beneficial since we can achieve numerous goals with the same robots, as opposed to ordinary robots that can only perform specific jobs. While Modular Robotic Systems (MRS) are expected to be employed in a wide range of environments, scales, and structures, their actual implementations fall short of their potential in performing real-world activities. For this reason, MRS solution approaches must be capable of resolving challenges arising from the interconnected

modules' tightly coupled kinematics, as well as their inherent resource, communication, and connection strength limitations.

The task of self-reconfiguration entails converting a vast swarm of modular robotic units from one shape to another. Three classes were generated from all prior self-reconfiguration approaches (Bottom-Up, Top-Down, and Theoretical) [10]. The target form is designed differently in each of these ways. Some use a target shape that is supplied at the start of the execution and adapt the algorithm to it, while others use a target shape that is created to fit the algorithm. The concept of adding the self-reconfiguration problem into MSR systems isn't new, and several solutions have focused on automating this process in order to allow the robot modules more freedom. Some findings lead to three abstract sorts of motions on the surface of a structure made up of many robots: linear motion, convex and concave transitions into another plane. A distributed locomotion control technique was suggested that can work with any system capable of one of those three motions [11]. They used a cellular automata (CA) model to create water-flow-like movement in this research, which evolves over a series of discrete time steps according to a set of rules dependent on the states of neighboring cells. The rules are then continuously implemented for as many time steps as are desired. Another study on self-reconfiguring robots that employed the cellular automata model focused on how to control the robots as well as the shape-reconfiguration process [12]. This procedure takes a 3D CAD model of the desired configuration as input and outputs a cellular automaton that represents that configuration. To boost efficiency even more, a gradient is utilized to steer the entire process while keeping the system connected at all times. A new approach for achieving global reconfigurable locomotion in MSRR modules with limited mobility [13] was presented by combining modular design with cellular principles. This strategy aided the robots in exploring new areas, avoiding obstacles, and adapting to them without experiencing any deadlock. Despite the fact that several CA-based systems have been developed and demonstrated to work, they have yet to be deployed on self-configuring systems. As a result, a next step in all of these studies is to figure out how to combine CA with MSRR to solve the self-reconfiguration problem.

In this paper, we focus on identifying and avoiding obstacles during the self reconfiguration and the shape-forming of a 2D self-reconfigurable robots. Our aim is to demonstrate the potential of MSRs to dynamically adapt to their environment and overcome obstacles in a shape transformation process. To do this, the system uses information about the robots' starting positions, target positions, and obstacle locations. The modules detect obstacles by analyzing their movement direction and future positions, alerting neighboring robots to coordinate a workaround and continue reconfiguration. This work fills a gap in research on obstacle recognition for 2D dynamic self-reconfigurable MSRs.

3 Self-Reconfiguration with Obstacle Detection

The robot's ability to self-reconfigure or self-assemble into a specific shape is one of the core difficulties of modular robot-based programmable matter. The distinction is that when a robot self-reconfigures, it is attempting to change the initial shape of the system into a new target shape. When a robot self-assembles, however, it implies it builds the shape from the ground up in order to assemble the required shape. Scalability, effective communication, and high target structure coverage are the essential characteristics of a self-reconfiguration method. In this paper, we are interested in modular robotic systems whose goal is to represent a certain object by self-reconfiguring from one shape to another after the system receives its description. However, the self-reconfiguration process is an incredibly complex task due to all of the possible ways the modules might be linked to each other. Additionally, self-reconfiguration is usually extremely slow due to the limitations imposed on module mobility, which make it extremely difficult to move a high number of modules simultaneously without significantly increasing the risk of module collisions or deadlocks [14]. Another issue with self-reconfiguration algorithms is the possibility that the robots will encounter a roadblock along the way, causing them to stop working and the system to shut down. We are concentrating our efforts in this paper on creating a first basic obstacle detection system for a 2D dynamic self-reconfiguration application. This method can assist a group of robots self-reconfigure in order to attain the desired shape while keeping an eye out for potential obstructions. Obstacles can be placed everywhere, even on the robots' paths and even in one of the target positions so it is up to the robots to self-reconfigure and adapt around the obstacles while always aiming to reach the target position.

Robots are expected to become a part of our daily lives in the future. They can act as a housekeeper, accountant, or whatever else we need assistance with to do some difficult and time-consuming chores. As a result of this context, many researchers focused on autonomous robots and their ability to self-reconfigure and interact with their surroundings. A robot is composed of sensors and actuators that allow it to communicate and move. Autonomous robot communications and their assessments of a given scenario are frequently inaccurate. This issue raised the topic of uncertainty, and many studies, such as Coalition Games with Uncertainty [7], have been conducted to address the uncertainty induced by noise sensors. However, inaccurate sensor data is not the only challenge. In an ideal world, robots would navigate obstacle-free environments, effortlessly reach their destinations, and accomplish their objectives without interruptions. Unfortunately, real-world environments are far from ideal. Robots often struggle to navigate obstacles, and even if they are trained to understand their surroundings, they must retrain and adapt whenever the environment changes. These challenges can cause delays in task execution or even system failures, presenting a critical hurdle in robotics. To overcome these issues, robots must be equipped with a higher level of autonomy and dynamic behavior, enabling them to adapt to environmental challenges and continue their tasks effectively. Our research addresses this need by developing a 2D self-reconfiguration algorithm.

3.1 Self-Reconfiguration Algorithm

In our approach, as we are interested in 2D self-reconfiguration modular robots, we consider the Hexanode robots, which operate on a hexagonal lattice and move by rotating around neighboring robots, either clockwise or counterclockwise. Our algorithm consists of five steps, outlined as follows:

- Step 1: Build the spanning tree in the system and calculate all the distances from the leader, root of the tree, to the leafs by sending distance messages from each robot to its neighbors.
- Step 2: Acknowledge the distance message, save the children of each module in a vector and send back the max distance of the robot in the subtree of this module as well as its ID. This will allow the root to get the Id and distance of the farthest node which will be the node to start moving.
- Step 3: Once the root has the Id and distance of the farthest node, send a message to the neighbors with the node Id to find the node that is supposed to start moving.
- Step 4: The node with the corresponding Id will stop the message broadcast and start moving in a clockwise position. The node will keep moving until it reaches the target with at least two neighbors or until it reaches the last target position it can reach. Then, the node would be set into position.
- Step 5: Once the node is set into position, it will become the new leader and apply the same previous steps (Step1 $\rightarrow$ Step5) to search for the farthest node from it so that it can move and join it as a target position and so on.

The algorithm stops once all the robots have reached their final positions. All these steps are detailed in Algorithm 1.

Algorithm 1: 2D Self-Reconfiguration Algorithm

$myDistance \leftarrow 0$
$myCurrentRound \leftarrow 0$
$isLeader \leftarrow false$
$myParent \leftarrow \emptyset$
$isInPosition \leftarrow false$
$myNbWaitedAnswers \leftarrow 0$
$myChildren \leftarrow \emptyset$
$maxDistanceNode \leftarrow 0$
$maxDistanceNodeId \leftarrow 0$

if $isLeader = true$ AND $isInPosition \neq true$ **then**
    $myDistance \leftarrow 0$
    $myCurrentRound \leftarrow 0$
    $isLeader \leftarrow false$
    for each $neighbor \in neighbors$ **do**
        send distance message to neighbor
        $myNbWaitedAnswers \leftarrow myNbWaitedAnswers + 1$

```
    end for
  end if

  if received a message then
    if message.type = distance then
      if myParent = ∅ then
        myDistance ← message.distance + 1
        myCurrentRound ← message.currentRound + 1
        neighbors ← neighbors - {message.origin}
        for each neighbor ∈ neighbors do
          send distance message to neighbor
          myNbWaitedAnswers ← myNbWaitedAnswers + 1
        end for
        if myNbWaitedAnswers = 0 then
          send acknowledge message to message.origin
        end if
      end if
    end if
    if message.type = acknowledge then
      myNbWaitedAnswers ← myNbWaitedAnswers - 1
      if message.parent = true then
        myChildren ← myChildren U {message.origin}
      end if
      if myNbWaitedAnswers = 0 then
        maxDistanceNode, maxDistanceNodeId ← getMaxDistanceNodeFromSubTree
        if myParent ≠ ∅ then
          send acknowledge message to myParent
        else
          for each child ∈ children do
            send leader message to child
          end for
        end if
      end if
    end if
    if message.type = leader then
      if module− > blockId = message.leaderId then
        isLeader ← true
        get the Hexanodes Motions
        Move the Hexanode to target
        myDistance ← 0
        myCurrentRound ← myCurrentRound + 1
        isLeader ← false
        myParent ← ∅
        isInPosition ← false
        myNbWaitedAnsers ← 0
```

```
            myChildren ← ∅
            maxDistanceNode ← 0
            maxDistanceNodeId ← 0
            for each neighbor ∈ neighbors do
                send distance message to neighbor
                myNbWaitedAnswers ← myNbWaitedAnswers + 1
            end for
        else
            for each child ∈ children do
                send leader message to child
            end for
        end if
    end if
end if
```

3.1.1 Obstacle Detection Algorithm

The major goal of this work is to assist robots in detecting any unknown object or impediment that may prevent them from accomplishing their tasks. Now that we have our basic 2D self-reconfiguration program, we modify our algorithm and include some additional information to help the robots recognize and adapt to obstacles. Therefore, we must verify at each robot movement that it is not bumping into an obstacle. What we do is, we check if the next position of a robot is an unknown object, then we stop the robot and we reinitialize all the values and we go back to the first step of the self-reconfiguration algorithm where we send a distance broadcast and we find the farthest node that will be the next one to move. Furthermore, in order to prevent the robots from becoming caught in a clockwise or anticlockwise loop, we check if they are stuck in a loop of movements that returns them to the same positions, and if they are, the direction of their movement is switched. The obstacle detection algorithm is presented in Algorithm 2.

Algorithm 2: Obstacle detection Algorithm

```
robotAtObstacle ← false
currentDirection ← motionDirectionStatic :: CW
if received a message then
    ...
    if message.type = leader then
        if module.initialPosition = module.nextPosition then
            currentDirection ← motionDirectionStatic :: CCW
        end if
        if module.nextPosition ∈ getWorld()− > obstacles then
```

```
            robotAtObstacle ← true
            myDistance ← 0
            myCurrentRound ← myCurrentRound + 1
            isLeader ← false
            myParent ← ∅
            isInPosition ← false
            myNbWaitedAnsers ← 0
            myChildren ← ∅
            maxDistanceNode ← 0
            maxDistanceNodeId ← 0
            for each neighbor ∈ neighbors do
                send distance message to neighbor
                myNbWaitedAnsewrs ← myNbWaitedAnsewrs + 1
            end for
        end if
        ...
    end if
end if
```

4 Simulation Results

In order to test the efficiency of our algorithm, we conducted several series of simulations using the simulator VisibleSim. The application requires a config.xml file as input, specifying the initial positions of the connected robots and their target positions. The next task is to implement the distributed block code algorithm, which orchestrates the movement of the robots to achieve the transition from Shape A to Shape B. We tested four scenarios while varying the configuration and the positions of the obstacles. In each of the four scenarios, the blocks, targets, and obstacles were defined, while varying the number of modules in the system. The complexity of the layouts ranges from a simple line of obstacles to a complex maze of obstacles.

4.1 Line of Obstacles

The first scenario consists of simulating our approach with a simple line of obstacles to ensure that the robots could identify and adapt to an obstruction with the use of our algorithm as shown in Figure 1. The robots' initial state is seen in grey in the first image, with their leader colored in yellow. All of the obstacles are in the shape of robots, but they do not work, thus they are classified as unknown objects and are colored purple. As seen in the second image, the robots were able to recognize the

obstruction and turn red to tell their neighbors to turn around it. The robots were able to build a new path to continue their mission by spinning around the obstructions. Finally, the robots reached their target places in the third image, with one robot remaining in red because the last target position was already taken by an obstacle. Without counting the walls (62 obstacles), this simulation used 14 robots and 5 main obstacles. During this self-reconfiguration, all of the modules performed a total of 239 moves and communicated 1271 messages with their neighbors within **17.719 seconds**.

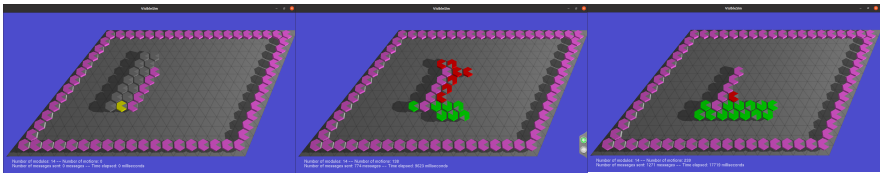

Fig. 1: Line of Obstacles Simulation Results

4.2 One line and Walls of Obstacles

To make things a little more difficult, we connected the line of barriers to the walls of obstacles on the map's perimeters. As a result, when the robots reach a wall, they are unable to get over the obstructions. We ran a simulation of this scenario, and the results are shown in Figure 2.

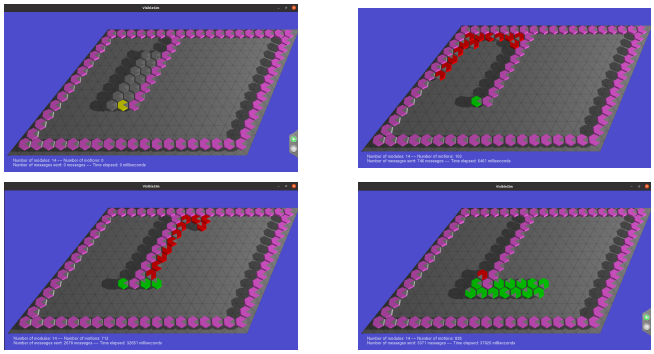

Fig. 2: Line and Wall of Obstacles Simulation Results

As previously stated, the robots are initially colored grey with a yellow leader, and the obstacles are purple. In this simulation, the robots encountered barriers while moving, but were unable to overcome them as in the prior example due to the fact

that they are connected directly to the wall. Based on our algorithm, the modules were able to find a method to get around the impediment by moving along the perimeter and adjacent to the four walls until they reached the other side. Because a barrier was located on a goal place, one robot was left out in this scenario as well. The same 14 robots, 9 primary obstacles, and barriers (62 obstacles) were employed in this simulation. All of the robots made a total of 835 actions while running, and 3371 messages were sent to their neighbors by all of them. For this instance, the entire form reconfiguration and obstacle detection process took **37.880** seconds.

4.3 Two Lines and Walls of Obstacles

In the previous scenario, we noticed how the modules used the walls as a guide to get to the other side. Therefore, by pressing them between the two lines of barriers, we planned to obstruct their path once more and observe how they would react to this new and more complex environment. The simulation results for this scenario are shown in Figure **??**. The robots are trapped between two lines of obstacles in this simulation, and while moving, they become stuck for a few seconds in a clockwise motion, attempting to find a way out. When they realized that moving clockwise was not taking them anywhere, they switched directions and began moving anticlockwise, which allowed them to discover a way out and arrive at their final objective. As previously, a red robot can't get anywhere since there's an impediment in the way. The 14 robots, 21 primary obstacles, and the walls (62 obstacles) were all employed in this simulation. All of the robots made a total of 513 actions while running, and they interacted with their neighbors with 13563 messages. For this instance, the entire form reconfiguration and obstacle detection process took **25.076** seconds.

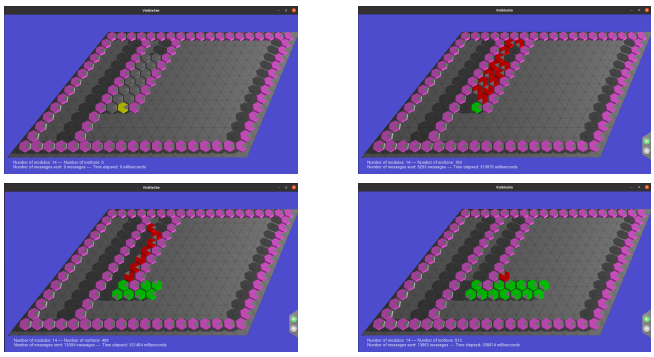

Fig. 3: Two Lines and Walls of Obstacles Simulation Results

4.4 Maze of Obstacles Simulation

In this phase of simulation, we decided to build a maze of obstacles between the starting point and the target position. That way, we'll be able to see how they perform in a large, unfamiliar environment and whether they'll be able to find a route out. We employed a maze generator created by keesiemeijer containing obstacles. This scenario is a little more complicated than the prior one as shown in Figure 4. The robots can be seen entering the maze and attempting to navigate it. In the second figure, we can see that they reach a zone with a dead-end at the end, forcing them to return to the entry point and take an alternative path. They run into the same problem in the fourth image, as they are looking for a dead-end section of the maze. However, the robots were always able to find a way out, and we noticed that they eventually reached the maze's finish and were able to take their final target locations. The maze was created with the help of 504 obstacles with 21 robots to navigate it. All of the robots made a total of 5389 moves while running, and 20904 messages were exchanged between them and their neighbors. For this situation, the entire form reconfiguration and obstacle detection process took 231.156 seconds, or 3.85 minutes.

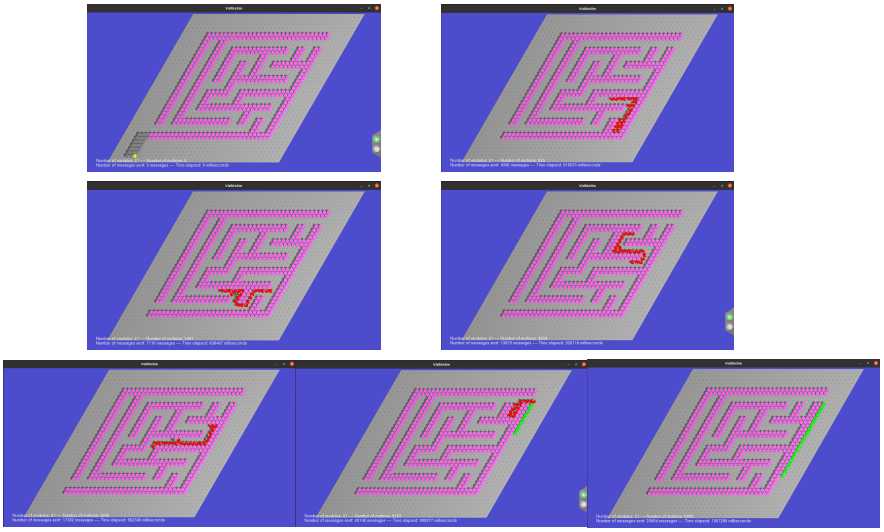

Fig. 4: Maze of Obstacles Simulation Results

We can summarize the results of the four simulations in the following table.

Table 1: Simulations Results Summary

Simulation	Nb Robots	Nb Obstacles	Nb Motions	Nb Messages	Time (sec)
Line Obstacles	14	5	239	1271	17.719
Line and Walls Obstacles	14	71	835	3371	37.880
Two Lines and Walls Obstacles	14	83	513	13563	25.076
Maze Obstacles	21	504	5389	20904	231.156

5 Conclusion and Future Work

This paper presents a 2D self-reconfigurable system capable of transforming itself from an initial shape to a goal one while addressing potential obstacles that could disturb the transformation. The proposed solution simulates the robots' movements, showing how they adapt to detected obstacles based on the provided initial and target shapes as well as obstacle positions. We evaluated the efficiency of our approach through four progressively challenging scenarios. In the first, a simple line of obstacles blocked the robots' path. In the second, the obstacles were connected to the walls, restricting the robots' ability to maneuver. The third scenario placed the robots between two parallel lines of obstacles, further complicating navigation. Finally, the fourth scenario involved a maze of obstacles requiring intricate navigation. In all cases, the robots successfully reached their targets by communicating with neighbors and adapting to the identified obstacles, even when suboptimal decisions led to longer routes.

As a future work, we will focus on extending the algorithm to 3D self-reconfigurable systems and enhancing its ability to handle uncertainties. This would involve developing mechanisms for detecting and adapting to unpredictable changes autonomously. Additionally, maintaining a heuristic approach will be critical to minimize reconfiguration time while optimizing the overall system performance.

Acknowledgment

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

References

1. Benoit Piranda and Julien Bourgeois. A Distributed Algorithm for Reconfiguration of Lattice-Based Modular Self-Reconfigurable Robots. pages 1–9. IEEE, February 2016.
2. Jad Bassil, Benoît Piranda, Abdallah Makhoul, and Julien Bourgeois. Asaps: asynchronous hybrid self-reconfiguration algorithm for porous modular robotic structures. *Autonomous Robots*, 48(7):16, 2024.
3. Jad Bassil, Jean-Paul A. Yaacoub, Benoît Piranda, Abdallah Makhoul, and Julien Bourgeois. Distributed shape recognition algorithm for lattice-based modular robots. In *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 85–91, 2023.
4. André Naz, Benoît Piranda, Julien Bourgeois, and Seth Copen Goldstein. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. In *Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on*, pages 254–263. IEEE, 2016.
5. Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007.
6. Jad Bassil, Benoît Piranda, Abdallah Makhoul, and Julien Bourgeois. Repost: distributed self-reconfiguration algorithm for modular robots based on porous structure. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12651–12658. IEEE, 2022.
7. Ramaekers Zachary, Dasgupta Prithviraj, Ufimtsev Vladimir, S. G. M. Hossain, and Nelson Carl. Self-reconfiguration in modular robots using coalition games with uncertainty.
8. Pierre Thalamy, Benoît Piranda, André Naz, and Julien Bourgeois. Visiblesim: A behavioral simulation framework for lattice modular robots. *Robotics and Autonomous Systems*, page 103913, 2021.
9. H. Bojinov, A. Casal, and T. Hogg. Emergent structures in modular self-reconfigurable robots. volume 2, pages 1734–1741. IEEE, 2000.
10. Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. A survey of autonomous self-reconfiguration methods for robot-based programmable matter. *Robotics and Autonomous Systems*, 120:103242, October 2019.
11. Zack Butler, Keith Kotay, Daniela Rus, and Kohji Tomita. Cellular Automata for Decentralized Control of Self-Recongurabl e Robots. *Proc. of the ICRA 2001 Workshop on Modular Robots*, July 2001.
12. K. Støy. Controlling self-reconfiguration using cellular automata and gradients. In *In Proc., 8th int. conf. on intelligent autonomous systems (IAS-8) (to appear*, pages 693–702, 2004.
13. Yanhe Zhu, Bie Dongyang, Sajid Iqbal, Xiaolu Wang, Gao Yongsheng, and Jie Zhao. A Simplified Approach to Realize Cellular Automata for UBot Modular Self-Reconfigurable Robots. *Journal of Intelligent & Robotic Systems*, 79:1–18, July 2014.
14. Pierre Thalamy, Benoit Piranda, and Julien Bourgeois. 3D Coating Self-assembly for modular robotic scaffolds. In *In Proceedings 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, 2020.