Learning system for self-reconfiguration of micro-robot networks

Baptiste Buchi

THÈSE DE DOCTORAT DE L'ÉTABLISSEMENT UNIVERSITÉ BOURGOGNE FRANCHE-COMTE

PRÉPARÉE À L'UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD (UTBM)

Ecole doctorale n°27

SPIM (Sciences Pour l'Ingénieur et Microtechniques)

Doctorat d'Informatique

Par

Monsieur Buchi Baptiste

Système d'apprentissage pour le problème d'auto-reconfiguration des réseaux de micro-robots

Thèse présentée et soutenue à Belfort, le 06/12/2023

Composition du Jury :

Mme, SCHARFF, Christelle	Pace University New York	Présidente du jury
M, CHARPILLET, François	INRIA	Rapporteur
M, EL GHAZAWI, Tarek	George Washington University	Rapporteur
M, ABOUAISSA, Hafid	Université de Haute-Alsace	Examinateur
M, GABER, Jaafar	Université de Bourgogne Franche-comté-UTBM	Directeur de thèse
M, MABED, Hakim	Université de Bourgogne Franche-comté-UFC	Co-directeur de thèse
M, LASSABE, Frédéric	Université de Bourgogne Franche-comté-UTBM	Co-encadrant de thèse

Declaration

I hereby declare this thesis represents my own work which has been done during my PhD degree at Université de Bourgogne Franche-Comté (UBFC / UTBM) with FEMTO-ST institute, and has not previously been included in a thesis or presented in any other institution for a degree or professional qualification.

I confirm that appropriate credit has been given within this thesis where reference has been made to the work of others.

The remaining contributions are proposed and carried by myself, where chapter 2, 3 and 4 have been submitted for publication.

Publications associated with this research

- Baptiste Buchi, Hakim Mabed, Frédéric Lassabe, Jaafar Gaber, and Wahabou Abdou. Translation based self reconfiguration algorithm for 6-lattice modular robots. In *2021 20th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 49–56, 2021. `https://ieeexplore.ieee.org/document/9521628`

- Francesco Witz, Baptiste Buchi, Hakim Mabed, Frédéric Lassabe, Jaafar Gaber, and Wahabou Abdou. Deep learning for the selection of the best modular robots self-reconfiguration algorithm. In *2022 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2022. `https://ieeexplore-ieee-org.ezproxy.utbm.fr/document/9912849`

- Baptiste Buchi. Système expert pour la sélection de l'algorithme d'auto-reconfiguration des robots modulaires. In *Ecole Jeunes Chercheuses et Chercheurs en Informatique Mathématique*, Maison de la Modélisation, de la Simulation et des Interactions. Nice, France, June 2022. `https://hal.science/hal-03709717v1`

- Baptiste Buchi; Hakim Mabed; Lassabe Frédéric; Jaafar Gaber. Machine Learning for Energy Efficient Modular Robots Self-Reconfiguration. In *The 20th IEEE International Conference on Ubiquitous Intelligence and Computing (UIC 2023)*, accepted, presented, to be published, Portsmouth, UK, August 2023.

- **SUBMITTED** - Baptiste Buchi; Hakim Mabed; Lassabe Frédéric; Jaafar Gaber. Machine Learning for Modular Robots Self-Reconfiguration Problem. In *Journal of Ambient Intelligence and Humanized Computing*

Acknowledgement

First of all, I would like to thank to all thesis committee members for the time they took to review my work and provide valuable feedback.

Then, I want to extend my profound appreciation and sincere gratitude to all individuals who have played pivotal roles in the realization of this doctoral endeavor. The research work presented here would not have been possible without the invaluable support, guidance, and encouragement received from my esteemed supervisors, Jaafar Gaber, Fredéric Lassabe, Hakim Mabed and Wahabou Abdou. They were instrumental in shaping the direction and depth of the research presented in this manuscript.

Gratitude is also extended to the Université de technologie de Belfort Montbéliard (UTBM) and the Université Bourgogne Franche-Comté (UBFC) who both welcomed me for the duration of my thesis and offered me a valuable environment in which made easier to work on this manuscript.

Futhermore, i want to take this opportunity to thank all other members of the OMNI team at FEMTO-ST for their interest and support at various point during those four years. I am grateful in particular to Julien Bourgeois and Benoît Piranda who are leading figures in the programmable matter research for taking time out of their certainly busy life to help me comprehend all the intricacies of this topic.

In addition, I would like to thank my enthusiastic fellow PhD students, Chafik Ahmed Amine especially, for their helpful advises and comments but also all the after-work, bowling nights and barbecues which have been a great help in relieving stress and fostering a positive workplace atmosphere. In addition, I am deeply appreciative of the financial support provided by the French Ministère de l'Enseignement Supérieur et de la Recherche, which has been crucial in facilitating the successful execution of this research project by founding my PhD.

Another person who has inspired me to pursue a career in scientific research is Christelle Scharff, to whom i am forever indebted to because she has sparked my interest for academic research.

Lastly, but most importantly, I am forever grateful to my family and friends for their unconditional love, unwavering encouragement, and constant belief in my abilities. Their unwavering support has been the driving force behind my perseverance and determination throughout this Ph.D. endeavor.

[N.B.: the rest of the acknowledgment is written in French.]

Je tiens à exprimer ma profonde gratitude et mes sincères remerciements à toutes les personnes qui ont contribué de manière essentielle à la réalisation de cette thèse. Ce travail de recherche n'aurait pas été possible sans le soutien inestimable, les conseils éclairés et l'encouragement constant de mes éminents directeurs de thèse, Jaafar Gaber, Frédéric Lassabe, Hakim Mabed et Wahabou Abdou. Leur mentorat éclairé, leur dévouement indéfectible et leurs commentaires pertinents ont été déterminants pour orienter la direction et la profondeur de cette recherche.

En outre, je tiens à exprimer mes chaleureux remerciements à mes collègues et aux autres chercheurs qui ont généreusement partagé leur expertise et des échanges stimulants tout au long de ce parcours académique. Les échanges d'idées et les efforts collaboratifs ont été enrichissants et inspirants, propulsant la recherche vers l'avant.

Enfin, mais surtout, je suis infiniment reconnaissant envers ma famille et mes amis pour leur

amour inconditionnel, leur encouragement constant et leur foi en mes capacités. Leur soutien indéfectible a été la force motrice de ma persévérance et de ma détermination tout au long de cette thèse.

L'achèvement de cette thèse marque l'aboutissement de plusieurs années de travail acharné et d'efforts soutenus, et je suis profondément reconnaissant pour les contributions de chacun qui ont rendu cette étape académique possible.

Thank You. Merci à vous.

Abstract

The problem of self-reconfiguration of micro-robot networks is one of the major challenges of modular robotics. A set of micro-robots connected by electromagnetic or mechanical links reorganize themselves in order to reach given target shapes. The self-reconfiguration problem is a complex problem for three reasons. First, the number of distinct configurations of a modular robot network is very high. Secondly, as the modules are free to move independently of each other, from each configuration it is possible to reach a very high number of other configurations. Thirdly and as a consequence of the previous point, the search space connecting two configurations is exponential which prevents the determination of the optimal schedule of the self-reconfiguration.

In this work, we propose, firstly, a distributed autonomous self-reconfiguration approach TBSR, focused on the optimization of movements for a better distribution of tasks. In other words, it involves distributing the effort made by each robot to reach the final shape.

Secondly, we propose hybrid approaches that take advantage of the advantages of centralized methods and distributed methods. These approaches make it possible to select the best distributed algorithm before launching the reconfiguration procedure. A range of distributed algorithms are pre-installed on each modular robot. At the start of the self-reconfiguration procedure, a coordinator broadcasts to all the micro-robots the data relating to the final shape to be achieved and the distributed algorithm.

To do this, we determined the relevant characteristics of self-reconfiguration problems allowing us to identify the most suitable algorithmic approach. A study of the impact of each reconfiguration method and performance parameters was conducted to establish a knowledge base. This database records the performance of various algorithms based on different parameters for a diverse range of self-reconfiguration problem scenarios.

Using a classification system, it is thus possible to establish for each self-reconfiguration method the characteristics of the self-reconfiguration scenarios for which it is effective. The learning mechanisms developed by AI (e.g., neural networks) are implemented. A first proposed hybrid CNNSR approach uses artificial neural networks to predict the optimal approach for self-reconfiguration. A CNN2SR approach (an improved version of CNNSR), was introduced for accuracy and error reduction, by refining the classification.

Thirdly, a modeling of energy consumption, resulting from real experiments with physical modular robots (Catom 2D) was established. This made it possible to implement a third hybrid CNN3SR approach focused on energy optimization for modular robots.

Keywords : Self-reconfiguration, Distributed algorithms, Learning, Modular robots, Artificial intelligence.

Résumé

Le problème d'auto-reconfiguration des réseaux de micro-robots est l'un des défis majeurs de la robotique modulaire. Un ensemble de micro-robots reliés par des liens électromagnétiques ou mécaniques se réorganisent afin d'atteindre des formes cibles données. Le problème d'auto-reconfiguration est un problème complexe pour trois raisons. Premièrement, le nombre de configurations distinctes d'un réseau de robots modulaires est très élevé. Deuxièmement, comme les modules sont libres de se mouvoir indépendamment les uns des autres, à partir de chaque configuration il est possible d'atteindre un nombre d'autres configurations lui aussi très élevé. Troisièmement et comme conséquence du précédent point, l'espace de recherche reliant deux configurations est exponentiel ce qui empêche la détermination du planning optimal de l'auto-reconfiguration.

Nous proposons dans ce travail, dans un premier temps, une approche d'auto-reconfiguration autonome distribuée TBSR, axée sur l'optimisation des déplacements pour une meilleure répartition des tâches. En d'autres termes, il s'agit de répartir l'effort fourni par chaque robot pour atteindre la forme finale.

Dans un deuxième temps, nous proposons des approches hybrides qui tirent profit des avantages des méthodes centralisées et des méthodes distribuées. Ces approches permettent de sélectionner le meilleur algorithme distribué avant le lancement de la procédure de reconfiguration. Une gamme d'algorithmes distribués sont préalablement installés sur chaque robot modulaire. Au début de la procédure d'auto-reconfiguration, un coordinateur diffuse à l'ensemble des micro-robots, les données relatives à la forme finale à atteindre et l'algorithme distribué.

Pour ce faire, nous avons déterminé les caractéristiques pertinentes des problèmes d'auto-reconfiguration permettant d'identifier l'approche algorithmique la plus adaptée. Une étude de l'impact de chaque méthode de reconfiguration et des paramètres de performances a été menée pour établir une base de connaissances. Cette base consigne les performances des divers algorithmes en fonction de différents paramètres pour un éventail varié de scénarios de problèmes d'auto-reconfiguration.

A l'aide d'un système de classification, il est ainsi possible d'établir pour chaque méthode d'auto-reconfiguration les caractéristiques des scénarios d'auto-reconfiguration pour lesquels elle se montre efficace. Les mécanismes d'apprentissage développés IA (e.g., réseaux de neurones) sont mis en œuvre. Une première approche hybride CNNSR proposée fait appel aux réseaux de neurones artificiels pour prédire l'approche optimale pour l'auto-reconfiguration. Une approche CNN2SR (une version améliorée de CNNSR), a été introduite pour la précision et la réduction des erreurs, en affinant la classification.

Dans un troisième temps, une modélisation de la consommation énergétique, issue d'expérimentations réelles avec des robots modulaires physiques (Catom 2D) a été établie. Cela a permis de mettre en œuvre une troisième approche hybride CNN3SR axé sur l'optimisation énergétique pour les robots modulaires.

Mots-clés : Auto-reconfiguration, Algorithmes distribués, Apprentissage, Robots modulaires, Intelligence artificielle.

Table of content

Contents

List of Abbreviations

C2SR: Cylindrical-Catoms Self-Reconfiguration (algorithm)

CNN: Convolutional neural network

IoT: Internet of Things

LOTG: List On The Ground

LOTL: List On The Left

MEMS: Microelectromechanical systems

ML: Machine Learning

MRR: Modular reconfigurable robots

MRSR: Modular robots self-reconfiguration

MSR: Modular self-reconfiguration

NN: Neural network

PM: Programmable matter

SR: Self-reconfiguration

TBSR: Translation-based self-reconfiguration (algorithm)

General Introduction

In this chapter, we state the context and the motivations and scope that drove the major contributions of this thesis. We synthesize the different conducted studies. Then we overview the organization of the manuscript.

Research topic

A programmable matter system is a collection of distributed modular components that can self-assemble, disassemble, and move. The modular robots act as elementary atoms in physical corps. Each module includes sensors, actuators, computational unit, memory capacity, and communication devices. The aim of such systems is to conceive dynamic self-adaptive robots or objects that fit a diversity of situations.

Research works concerning programmable matter focus globally on two main challenges. The first one refers to the conception of the modular robot hardware taking into account the stability of the links, the reconfiguration responsiveness, energy requirements, and computational capabilities. The second challenge concerns the conception of software solutions that allow a high number of modular robots to cooperate in order to reach a given targeted configuration or shape.

This thesis addresses the second challenge by introducing an original approach for the Modular Robots Self-Reconfiguration (MRSR). The idea is to exploit the proposed methods in the literature rather than propose a new method. Indeed, the recent literature on MRSR problem shows a variety of methods differing in their strategy and the targeted hardware. These works lake of performance analysis and comparison with the previous methods, which makes it difficult to understand where (which hardware) and when (which problem instance) a given method is efficient. We propose to use Machine Learning techniques to identify on the basis of the MRSR problem formulation which algorithm is expected to produce the best results in terms of success rate, convergence time, and energy consumption.

1 Problem statement

MRSR problem is formulated as a set of modular robots with just a local knowledge of their environment that cooperate to form the final shape or organization. In this thesis, we address the 2D modular robot systems. This means that the set of robots moves within a horizontal plane like a table or a vertical plane over a ground support.

Even if there are two main approaches for MRSR: distributed and centralized, the centralized approach cannot manage a huge number of modular robots. In contrast, the distributed approach is based on the execution of the same algorithm over each modular robot. Therefore, modular robots use local rules (based on local knowledge and impacting the local area) to determine future actions. The global algorithm behavior is an emergence of the parallel action of the individual robots. The distributed approach fits the capacity limitation of the modular robots. However, the efficiency of the algorithm in front of a specific problem instance is less predictable.

This thesis aims to develop a hybrid approach allying the advantages of both centralized and

distributed approaches. The objective is to use a centralized pre-processing mechanism based of Artificial Intelligence (IA) and Artificial Neural Network (ANN) to select the most suitable distributed algorithm to run over the modular robots. This way, the unpredictability effect of the distributed approaches is mitigated. More precisely, the goal of this thesis is to develop a self-reconfiguration AI-based method that can for each self-re-configurable scenario predict the best approach to be used by the modular robots.

2 Contributions of the thesis

- **Design of a new distributed approach for self-reconfiguration:** In this first part, we propose a distributed asynchronous self-reconfiguration approach (TBSR) allowing to distribute the effort made by each robot to reach the final shape. This makes it possible to extend the battery life of the micro-robot network.
- **Creation of an Artificial Neural Network system to predict the best-suited approach for self-reconfiguration:** In this second part, we present a pre-processing algorithm, called CNNSR, using a neural network technique for selecting the best distributed MRSR algorithm. The objective is to propose a centralized procedure that allows, depending on the self-reconfiguration problem, to determine which algorithm is the most suitable, in terms of complexity.
- **Fine-grained improved AI-based approach:** In this part, we present an advanced working scheme for the CNNSR, called CNNSR2, for the prediction of the best-suited distributed MRSR algorithm. We design this method to be a fine-grained alternative, allowing us not only to be more precise but also to greatly reduce the impact of the potential mistakes.
- **Energy movement and action optimization for modular robots:** In this section, we will introduce a new system that we designed to improve the realism of self-reconfiguration simulation with a focus on energy consumption based on real-life data and previous tests. Then we will compare previous approaches using this system before introducing a new AI-based method to select the best approach when a deep look at the energy consumption of these approaches is in place.

3 Title of the thesis

(In English) Learning system for self-reconfiguration of micro-robots networks.
(In French) Système d'apprentissage pour le problème d'auto-reconfiguration des réseaux de micro-robots.

4 Organization of the thesis

4.1 General presentation

This manuscript is organized into four parts. The first part is a general introduction to programmable matter and self-reconfiguration where we state the different shape that can take programmable matter, the approaches used to implement them and the methods needed to interact with.

The second part describes a new approach for self-reconfiguration based on translation for network of modular robots in a hexagonal grid. Here, we will look at this method in the following

way : i) Space modeling ii) Algorithmic functioning iii) Experimental results iv) Potential upgrades.

The third part presents three proposed approaches based on neural networks to predict the best suited distributed self-reconfiguration method for 2d modular robot network in a hexagonal grid. A chapter will go over each approach following the same steps : i) presenting technically the considered solution, ii) description of the neural network used, iii) analysing the experimental results and iv) discussing potential upgrades.

The fourth part provides a summary of the presented contributions, analyses their distinctions and potential application. This part is also where we provides insights for future approaches.

4.2 Outline : global overview

- General Introduction
- Chapter I : State of the Art
    1. Modular Robots
    2. Self-Reconfiguration challenge
    3. Modular Reconfigurable Robots simulators
- Chapter II : Translation-Based Self-Reconfiguration Approach
    4. Translation based self-reconfigurable algorithm
    5. Experimental results of TBSR
- Chapter III : Artificial intelligence based approaches
    6. Hybrid approach powered by artificial intelligence
    7. Fine grained artificial intelligence approach
- Chapter IV : Energy-aware approaches
    8. Energy cost model
    9. Artificial neural network for CNN3SR
- Chapter V : Conclusions and perspectives
    10. Conclusions
    11. Perspectives

5 Content of the chapters

I. State of the Art

This chapter is an introduction to programmable matter, self-reconfiguration problem and modular robots and will shows the context of our specific works, which robot platform was used, which self-reconfiguration simulator was chosen and explain all of those choices. It is in this chapter that we will look at the previous work that exist on distributed algorithm, programmable matter and self-reconfiguration.

II. Translation based self-reconfigurable approach

This chapter describes an original distributed self reconfiguration approach, TBSR, based on the principal of mass translation. The objective is to increase the stability of the left to right distribution of the modular robots over the initial shape and the final built shape. We then compare TBSR with a rival algorithm that can be used for the same scenarios.

III. Artificial intelligence based approaches

In this chapter, we will do a state of the art on artificial intelligence and more specifically on the methods that were considered or used in the following chapters like multi-layer perceptron or convolutional neural network.Then, the chapter will deal with the creation of neural networks used to predict the right self-reconfigurable class of a scenario.

III.6 Hybrid approach powered by artificial intelligence

We used a first hybrid centralized/distributed modular robots reconfiguration approach, CNNSR. With this approach, a convolution neural network system is used to estimate the most adapted distributed reconfiguration algorithm according to the initial shape formed by the modular robots and the target shape. Two distributed algorithms are studied: C2SR and TBSR. The designed CNN model allows determining which option is the best for a given reconfiguration problem.
We applied the Neural Network technique to two self-reconfiguration algorithms: C2SR and TBSR. The obtained results show that the ML tool succeeds 96.67% of the time to determine the suitable algorithm based on the initial and the final shape. Consequently, using ML leads to the reduction of the required number of moves for the reconfiguration.

III.7 Fine grained artificial intelligence approach

We designed a second hybrid approach, CNN2SR, a fine-grained artificial intelligence model to predict the best suited approach for self reconfiguration and reduce the risk of the worst outcomes (when the model don't choose the best approach and choose the worst instead) to appear.
To do so, we use the data created for the previous chapter and add granularity to it as well as an overall redesign of the Convolution neural network used before to improve it's performance, with a new precision of 97,25%.

IV. Energy-aware based approach

This chapter is where we propose a tool that can help choosing the algorithm the best suited to the situation based on the level of energy in each robots. To do so, we have add a complex energy resource system to our simulation based on previous research and real life data. The new model is able to select the best suited approach 97.88% when we filter the approach by their energy consumption.

V. Conclusions and perspectives

This chapter will conclude and look at the perspectives of the thesis, what could be done to expand the work done here.

Chapter I
State of the Art

This chapter gives a general introduction to the programmable matter, starting from the motivations behind this technology and its early conceptualizations. Then we delve into various implementations proposed in this field, examining different approaches and devices. Furthermore, we discuss the strengths and weaknesses of the technologies employed.

1 Introduction

Since the dawn of humanity, we tried to use materials around us to improve our lives. At first, each tool was only used once before being abandoned until we quickly realised the potential of keeping them. The time not wasted looking for a new tool for an already solved problem, we can spare finding better alternatives to improve our toolbox. All that leads to the creation of better tools that give us more options when interacting with the previous materials and path to discover new ones, which in turn can end with better tools.

As time went on, we developed many techniques and learned about our world, which allowed us to access more reliable and efficient processes to model, form, and deform the matter. With the rise of automation in the last century, research on this topic has resulted in many new complex operations that shape the matter.

As our control over matter evolved, we realized not only that most of the processes generate waste, materials not needed at the end of our work, but that we can use our knowledge and our tools to use those materials for different goals. If we can change the matter around us and recycle previously unused materials, a new question appears: Can we create a matter that will always be useful? A modeling clay that isn't restricted to only being used as a prototyping tool but that can be used efficiently. We call this technology *programmable matter* (PM).

The programmable matter was first defined by Toffoli and Margolus in 1991[99] before Goldstein and Mowry gave a more generalized definition in 2005 [30, 29]. This is how they describe it as: "[...] a technology that will allow one to control and manipulate three-dimensional physical artifacts"[27].

A product made out of programmable matter will be able to change its shape, properties, and features, allowing us to have incredible tools that can adapt themselves to many situations. But how to make such material? At first, chemistry, cognitive science, molecular computation, and robotics were looked at, with the intent of using biological processes to achieve such progress. In order to do so, the academic research focused on embedding molecules with instruction to spontaneously assemble into complex structures[48]. The goal was to obtain a matter that can have the following properties: Evolutivity (be able to change its shape), Programmable (the change of shape is driven by an externally controlled stimulus), Autonomy (the matter doesn't need external help to achieve the shape-shifting) and Interactivity (the matter can give feedback to the user)[12].

During the last fifty years, miniaturization has been a big trend in computer science. Smaller and smaller components make it easier to build micro-robots than ever before, allowing us to move forward on the path to create programmable matter. The idea of programmable matter can appear far-fetched but when we look at the current trends in technology, we can see it tends towards a programmable matter environment, especially with the rise of the Internet of Things

(IoT), or even the Internet of Everything (IoE).

While many technologies exist for programmable matter, the only one that gathers all those properties is Modular Re-Configurable Robots (MRR) [104]. In this case, the programmable matter is then composed of a set of modular robots that can interact with the other modular robots and change the global organization of the system. With that in mind, MRR, with distributed algorithms especially, soon became the most widely studied branch of programmable matter research. It is now more popular than other programmable matter branch such as biological methods[47], 4d printings [6] & smart materials [15] or quantum wellstone[65].

In this thesis, we focus on the use of MRR systems as a way to reach the programmable matter objective.

The self-reconfiguration problem of MRR system, called MRRSR, consists of defining the software solution that allows changing the organization of the modular robots. It could be seen as a specific application of the Optimal Transport within the robotic field: efficiently move a distribution of mass, such as a stack or, in our context, a swarm of robots, to another place [9], [51],[3].

More formally, the MRRSR problem is defined by the current shape of the modular robots and the desired shape to achieve.

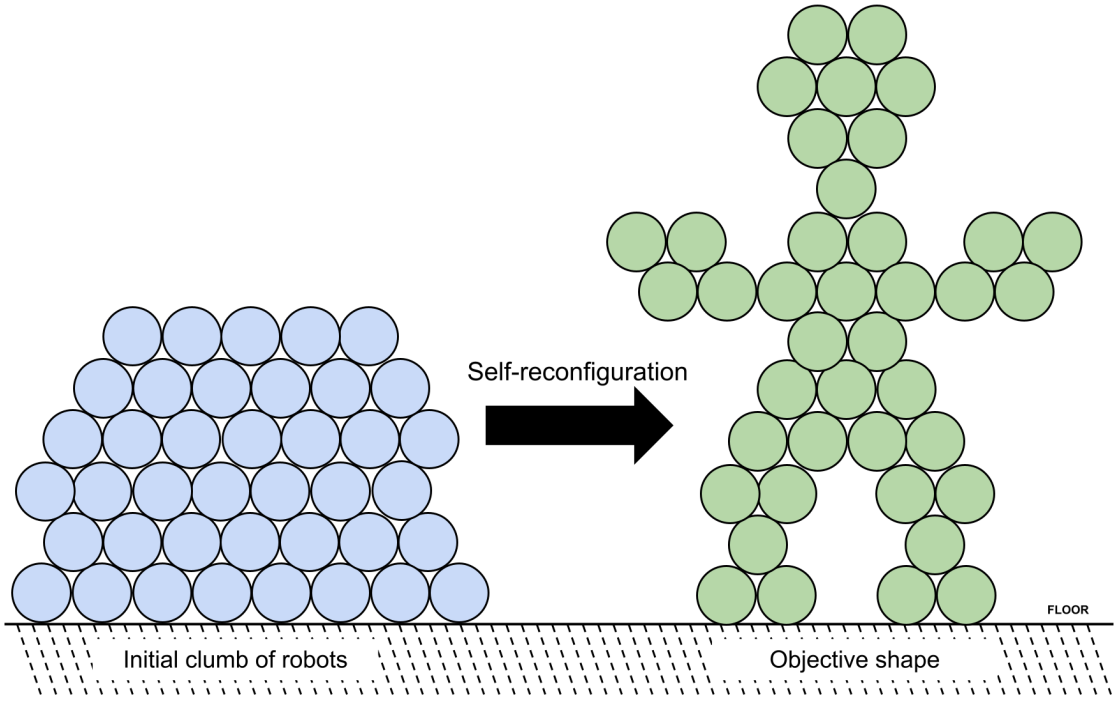


Figure 1: Application scenario of programmable matter in a 6-lattice in 2 dimensions.

2 Modular Robots

2.1 Modular Robotic Platforms

The concept of modular robotics was first introduced in the late 80's as a cellular robotic system by Fukuda and Kawauchi [24]. Modular robotics is a robotic system composed of identical interconnected elements called micro-robots, modular robots, or modules. Each modular robot is characterized by the ability to move around the others in order to constitute a variety of shapes or topologies. Kasper Støy and Haruhisa Kurokawa [94] define modular robots as a robotic system comprised of interconnected electro-mechanical modules capable of re-configuring themselves to adapt optimally to their task and/or environment or recover from failures.

In 2002, Seth Goldstein and Todd Mowry started the Claytronics project at Carnegie Mellon University. This research endeavor combines principles from modular robotics, systems nanotechnology, and computer science, culminating in the realization of a dynamic, three-dimensional display of electronic information referred to as "claytronics". Since 2016, the Femto-ST Institute has taken the lead on the project [4]. The Claytronics project specified two main objectives to reach that goal :

- Creating the basic modular building block of the programmable matter called claytronic atom or **catom**.
- Designing and writing robust and reliable software programs that will manage the shaping of ensembles of millions of catoms into dynamic, 3-Dimensional forms.

Many other architectures have been proposed over the years for modular robotics [5]. However, swarm robotic systems are not considered as modular robots systems [34]. The most fundamental difference between modular robots and conventional swarm robots is the virtue of each individual robot, or module, to remain continuously interconnected within the system. The connectivity of modular robots serves as a crucial basis for communication, synchronization, and power distribution among the modules. In contrast, swarm robots typically possess complete autonomy in terms of power and mobility, although certain systems are occasionally denoted as *mobile* modular robotic systems, such as Kilobot [85].

Within the Claytronics project, a team of researchers has successfully tested and validated millimeter-scale cylindrical catoms, which exhibit electrostatic actuation and self-contained functionality. As a simplified initial approach, the researchers opted to construct cylindrical catoms, called "**2D Catoms**", instead of spherical ones [45]. Figure 2 shows the 2D Catom prototype. The 2D Catoms have undergone partial validation through the fabrication of a hardware prototype. Each 2D Catom is composed of a cylindrical shell measuring 6 mm in length and 1 mm in diameter. A high-voltage Complementary metal–oxide–semiconductor (CMOS) die is affixed within the tube, encompassing a storage capacitor and a basic logic unit. The tube employs electrodes for power transmission, communication, and actuation. In the current design, a 2D Catom is capable of rolling on a power grid. Theoretical analysis indicates that a 2D Catom has the potential to complete a full revolution in either 1.67 seconds or 3.35 seconds.

2.2 Modular Robots classification

Some researchers consider modular robotic systems as tightly linked swarm robotics systems. However, the connectivity constraint impacts strongly the modular robots reconfiguration schemes, which makes modular robots a separate system case [34, 35, 40].

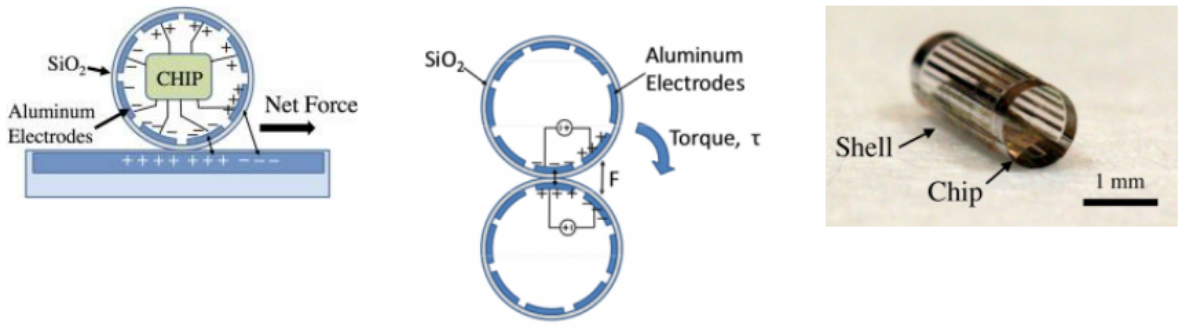


Figure 2: 2D Catoms schematics.

Each modular robotic system usually falls under one of two architectural categories. The first one is the chain-type modular robots where a structure is constructed by forming chains of interconnected modules, arranged in a tree-like manner. These modules function as joints or links and possess limited degrees of freedom (DOF). By assembling together, they collectively contribute to the overall high DOF of the structure. Polybot [103], CONRO [14] and YaMoR [67] are chain-like modular robot platforms. The second architectural category is the lattice-type modular robots. In such architecture, the modular system is composed of an ordered arrangement of modules, residing on a regular structure named a lattice. Modeling of lattice-based modular robots is easier due to the discretization of the module positions. Besides, they can be more conveniently controlled in parallel compared to chain-like modular robots. Various lattice structures exist, each based on specific cellular geometries, differing in packing density, dimensions, and the number of neighboring positions at a given location. These attributes have been explored in studies by Naz et al. [72] and Piranda et al. [81]. ATRON [42], M-Block [84] and Claytronics[33] are lattice-like modular robotic platforms.

Some modular robots present characteristics of both lattice-type and chain-type modular robots, and they are commonly referred to as hybrid modular robots. M-TRAN [43], SMORES [41] and Superbot [88] are several examples of such systems. Finally, there are a few exceptions of modular robots that do not precisely belong to none of these architectures. These cases, such as the FireAnt system [96], lie outside the scope of the present manuscript.

Other criteria may be used to establish more accurate classification of the numerous modular robots platforms proposed in the literature [103, 8, 41, 84, 82, 7]. Hereafter a set of three criteria related to the modular robots' organizational capabilities.

- **compact robots vs. mobile or swarm robots**: This feature relates to the presence or the absence of physical contact between the robots. Kilobot [85] is an example of a contactless swarm robots platform. The kilobots move using a vibration motor and communicate using reflected infrared light. In contact-based robots, the robots are linked to each other in a static or dynamic way. As we said above, the swarm robot systems may be excluded from the modular robot systems due to the absence of connectivity constraints.
- **individual vs. collective motion**: The modular robots can be distinguished according to their motion nature into three types. In the first class such as [85, 84, 82, 1], the robot's move may constraint the move of neighboring robots but the robot's motion is the result of its own individual action and does not impact the position of the other robots. In the second class of modular robot motion [8, 41, 42], the individual actions of a given robot can impact the position of one or several robots. A typical example is the arm-like

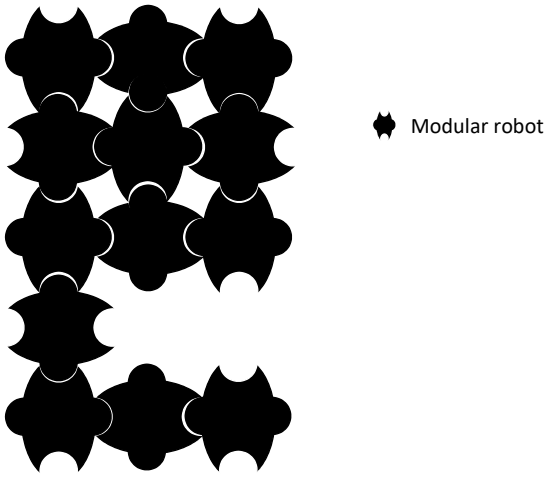
Platform	compact	regularity	motion type	topology
ATRON[42]	yes	yes	global impact	dynamic
MTRAN3[43]	yes	yes	global impact	dynamic
Polybot[103]	yes	no	global impact	fix
Roombot[8]	yes	yes	global impact	dynamic
Smores[41]	yes	yes	global impact	fix
M-Blocks[84]	yes	yes	local impact	dynamic
RoboGen[7]	yes	no	global impact	fix
SmartBlocks[82]	yes	yes	local impact	fix
2D-Catom [1]	yes	yes	local impact	dynamic
Catom(Claytronics)[28]	yes	yes	local impact	dynamic
Datoms[80]	yes	yes	global impact	dynamic
Kilobot [85]	no	no	local impact	dynamic
TERMES system [21]	no	yes	local impact	dynamic

Table 1: Characteristics of different major robots platforms.

modular robots [59] where the rotation of the robots forming the elbow leads to the lifting or lowering of the robots forming the hand. The third class of modular robots represents the case where the robots' motions need synchronization and cooperation between the robots to perform a collective move. For instance, in SmartBlocks system [82], a set of modules, composing a column or a line, are synchronized before shifting together.

- **Fixed vs. dynamic neighboring topology**: This criterion relates to the ability of each module to change its neighbors. In fixed neighboring topology, each robot keeps the same neighbors, even if the relative positions of the neighbors change. In dynamic neighboring topology, the robots are able to dock/undock each other dynamically.

Figure 3: 4-lattice modular robots organization.



In table 1 we summarize some of the major modular robot platforms and specify their characteristics.

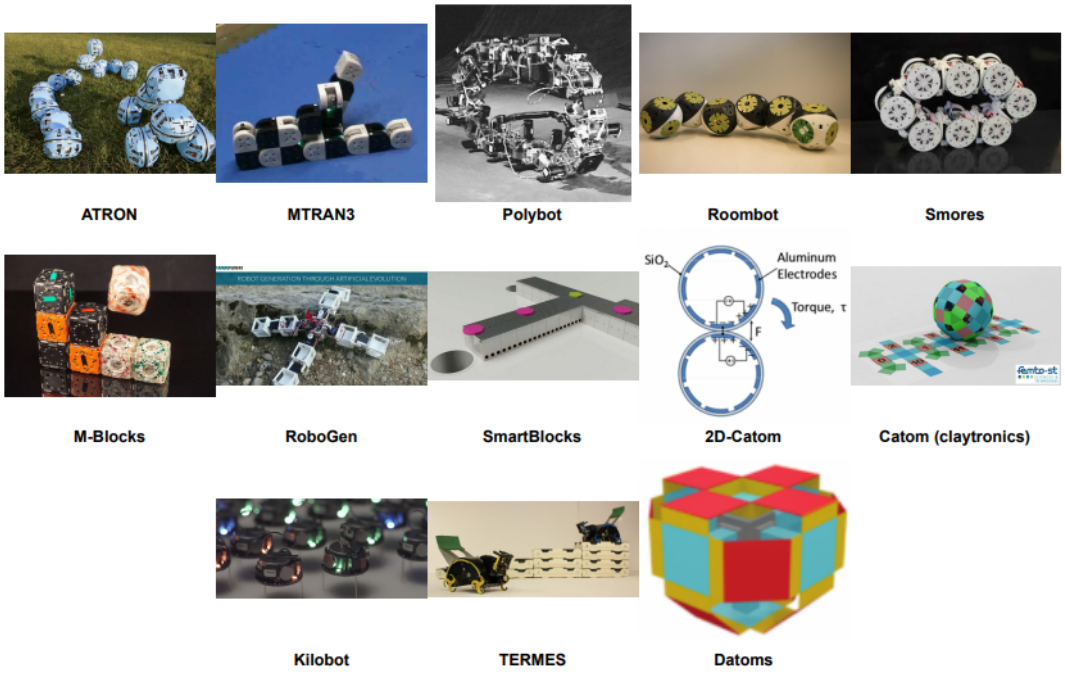


Figure 4: examples of modular robots.

3 Self-reconfiguration challenge

The conception of the Self-Reconfiguration algorithm for Modular robots (MRSR) is a very challenging problem. Indeed, the combinatorial complexity of the problem is enormous. An MRR networwork can be modeled by a graph $G = (V, E)$ where the vertices of G ($V = (v_0, v_1, ..., v_n)$) represent the modules while the edges of G ($E = (e_1, e_2, ...e_c)$) are the connections (neighboring relationships) between the modules[77], with n the number of modules in the network and c the number of connections between the modules.

The mass translation corresponds to the collection of moves required to move from one network representation G_1 to another network representation G_2. The mass translation is noticed as a function $f : G \rightarrow G$. The number of unique network configurations with n modules is $2^{w \times n}$ where w is the maximum number of simultaneous connections per module. Therefore, if the translation ofG_1 to G_2 requires at least s rounds, the algorithm should then find the optimal translation among $2^{w \times n \times s}$

The sub-optimality of the mass translation solutions found by the Self-reconfiguration algorithms is not just a waste of time, but also a waste of energy that represents a serious threat to the modular robots' lifespan. Therefore, conceiving efficient self-reconfiguration algorithms is a determining factor in the success of a modular robot platform.

Few works in the literature tried to model the MRSR problem by mathematical models. In [31], the MRSR problem was modeled by the propositional satisfiability problem (PSAT). The proposed model offers many advantages: PSAT is one of the most studied NP-complete problems; many exact and approximation-solving approaches were proposed and the proposed model is relatively complete. However, the proposed model is dedicated to the metamorphic modular robots system [17] and ignores the connectivity constraint of the system. In [61], the MRSR

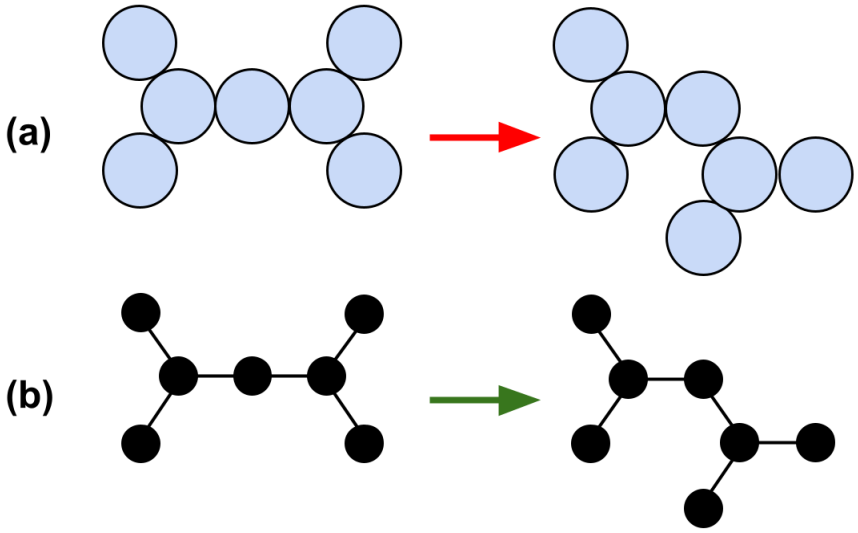


Figure 5: Graph representation is not fit the shape-shifting problem as in (b) both graph are identical but the network shapes in (a) are not the same.

problem is modeled by a Linear Programming system. The model provides a first lower bound that allows estimating the minimum number of moves needed to reach a target shape starting from an initial shape. However, the proposed LP model does not take into account some synchronization, connectivity, and friction constraints. In [57], a relaxed version of the problem (the modular robots are regrouped into a porous meta-module grid) is modeled as a kind of successive max-flow problem. The problem is then formulated as the conjunction of two problems:

- association problem consisting of mapping each a starting position in the initial shape with a destination position in the final shape
- max flow problem consisting of determining how the modular robots are routed in a competitive way to their respective destination positions with respect to the physical constraints between modular robots.

The proposed approach suffers from two main disadvantages. First, the authors have introduced a specific modular robot platform, where some robots are fixed and form a porous grid. Secondly, the robot's motion ignores the friction between robots.

In Table 2, we summarize the characteristics of the different proposed approaches in the literature. The features of the proposed MIQP-based self-reconfiguration method are given in the last row. From this table, we deduce that deterministic approaches are efficient when the initial and the final shapes respect the feasibility condition of the approach. However such approaches are difficult to extend to all cases. Conversely, multi-agent approaches are more flexible but suffer from random convergence performances: number of moves, reconfiguration delay, etc. Centralized approaches (similarity-based and optimality-based) use constraints relaxation to manage the MRSR problem. This relaxation makes the solved problem not realistic and therefore limits the relevance of the provided reconfiguration plan. The memory, algorithmic, and communication complexities measure the effort required by modular robots. The last column of Table 2 refers to the ability of the approach to determine if a MRSR problem is feasible or not.

Table 2: Comparison of Approaches

Method	Deterministic Distributed approaches [70][13]	Multi-agents approaches [106, 56, 102]	Similarity based approaches [16, 55]	Optimality based approaches [61, 57, 31]	Our Approach
Realistic	✓	✓	×	×	✓
Precalculated plan	×	×	✓	✓	✓
Convergence	Near-optimal	Random	Random	Random	Near-optimal
Spatial complexity	Low	Low	Low	Low	Low
Algorithmic complexity	Average	High	Low	Low	Low
Communication complexity	High	High	Low	Low	Low
Constraints on the shapes	Hard	No	No	No	No
Feasibility prediction	×	×	×	✓	✓

4 Modular Reconfigurable Robots simulators

In order to help in the conception of efficient MRSR algorithms, some works address the conception of a realistic physical and programming simulation environment for modular robots. VisibleSim [79] is a Modular Robot simulator developed by OMNI team from the FEMTO-ST Lab. This tool allows to simulation and programming of several MRR systems such as 2d Catom, Blinky Blocks, and 3D Catom.

Other simulators are proposed and can be used to simulate distributed MRSR running over robotic modules. ReBots simulator [19] is another efficient simulator that includes the simulation of manipulation and transportation applications using modular robots. The simulator includes the simulation of passive objects representing the transported or manipulated objects. ReRobots was successfully used to simulate and validate SuperBots and RoomBots programs. We can also cite Sim [19], another modular robots simulator offering a lightweight open-source simulator for modular robot applications.

Our studies illustrated hereafter concern the 2D Catoms system. Simulations are conducted using VisibleSim tool. However, all the concepts and methodologies described in this manuscript can be extended with other modular robot platforms.

VisibleSim is a framework tailored to researchers. This C++ framework facilitates the construction of simulators for lattice-based modular robots that are orchestrated through distributed programming paradigms. Notably, VisibleSim is equipped with multiple exemplar modular robot simulators packaged within the software. It is imperative to note that VisibleSim is offered as an open-source project subject to the terms of the AGPLv3 license and is made accessible through the Github platform.

In order to facilitate the creation of new configuration for self-reconfiguration in a 6-lattice grid, we developed a tool that allow us to quickly create new mass translation scenarios visually. The

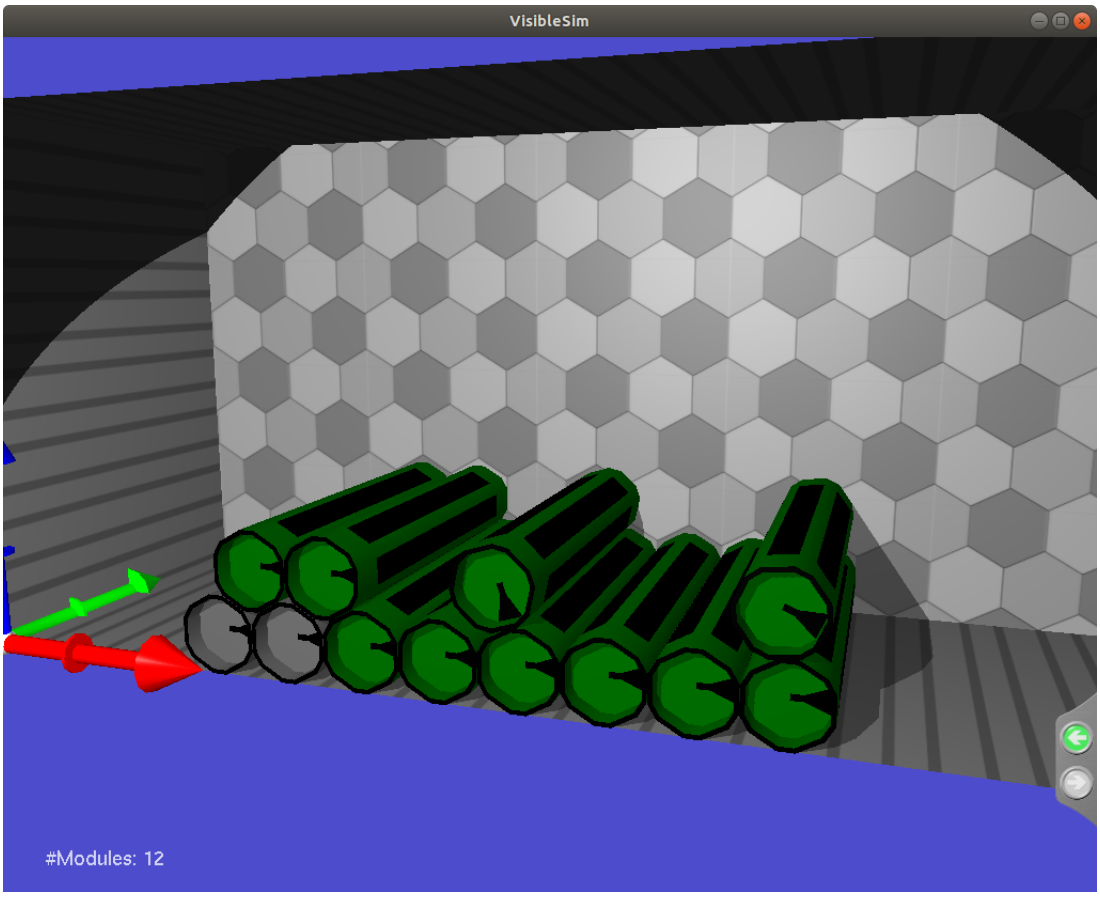


Figure 6: Screenshot of VisibleSim.

different scenario can be save as file for the script which can load them later and is compatible with multpiple format such as CSV, XML or XLSX. The tool can also fuse, convert and do a variety of other operation on each file, making it easier to create new variation of existing scenarios. The tool was made in python and called VisibleDraw, it is available through the Github platform. That tool was heavily use for the creation of the multiple datasets presented in this study.

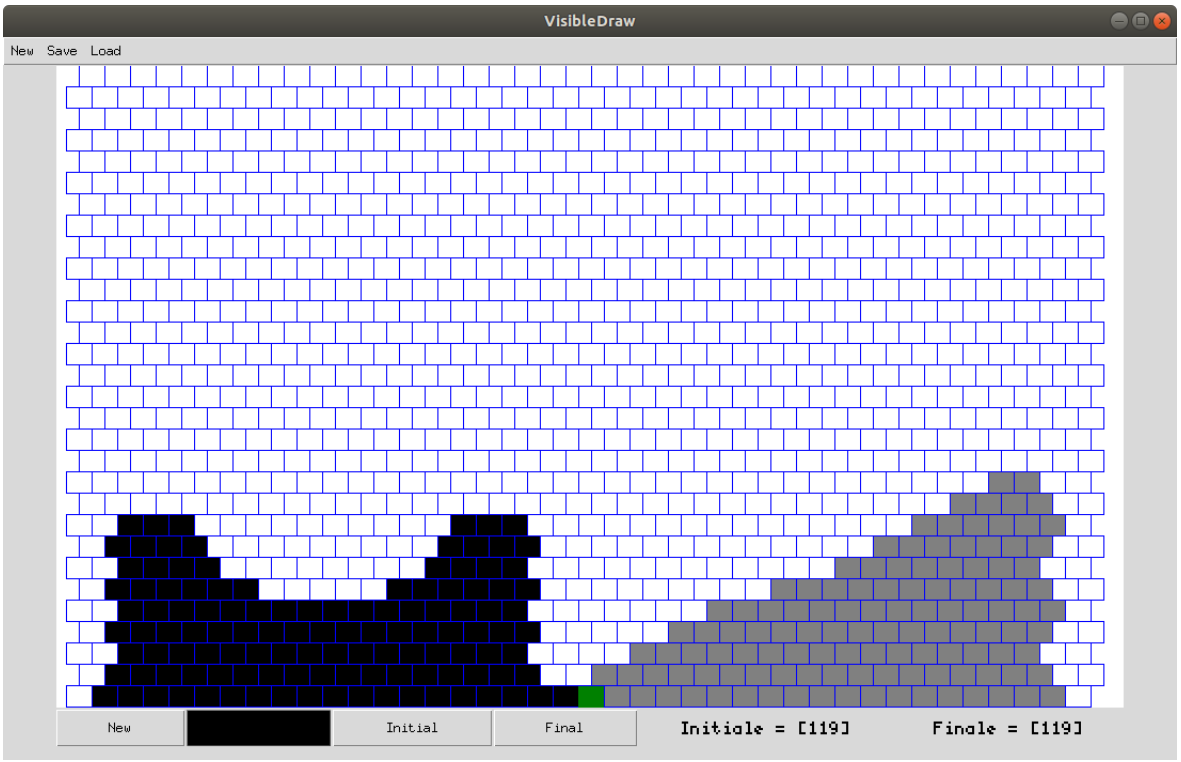


Figure 7: Screenshot of VisibleDraw with the Vase to Slope 1 (large size) scenario open.

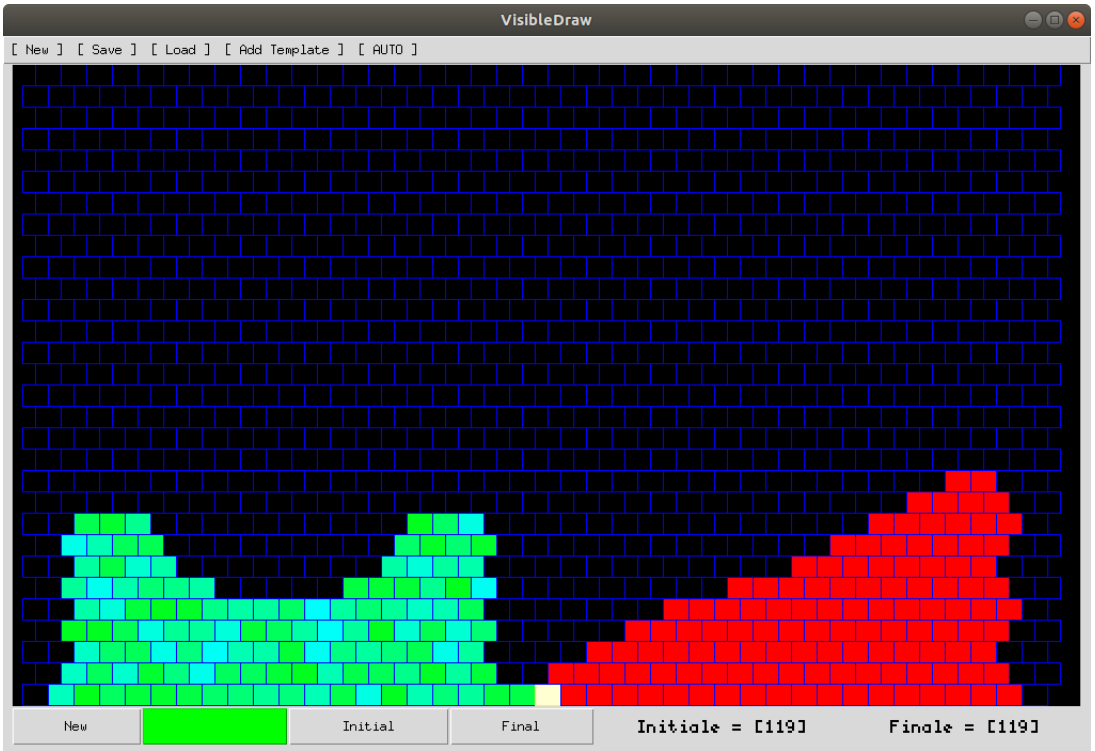


Figure 8: Screenshot of VisibleDraw on energy simulation mode with the Vase to Slope 1 (large size) scenario open with template 10 (random energy).

Chapter II
Translation-Based Self-Reconfiguration Approach

Our objective in this thesis is the elaboration of preprocessing mechanism that helps the modular robots in the selection of the most suitable self-reconfiguration algorithm according to the faced problem. To reach this objective, it is important that modules have more than one distributed algorithm from which to choose. The diversity of MRR platforms makes it difficult to find different self-reconfiguration algorithms used for the same MRR system with the same constraints. For instance, the algorithm of Bateau et al. [10] is conceived for 6-lattice cylindrical robots moving within a horizontal plane such as a table. Therefore, a modular robot can bypass a block of modules by both sides (see figure 9). In addition, there are no gravity constraints that impose a given order of construction. However, for 2D Catoms system, modules are piled up on top of each other starting from the ground level. Therefore, a module cannot reach a given position if the above positions are not filled.

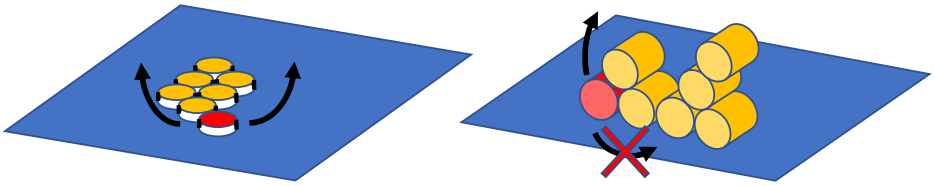


Figure 9: Difference between 6-lattice horizontal catom system (left) and Cylindrical 2d Catom (right).

In this manuscript, we selected the 2D Catom system as a case study. Only one published algorithm, called C2SR, was proposed for such a system. Since our objective is to provide a solution that allows to determine which algorithm is better for a given MRSR problem, conceiving a challenging algorithm is a priority. We describe, in this chapter, an original distributed self-reconfiguration approach, TBSR (Translation-based Self-Reconfiguration), based on the principle of mass translation. The objective is to increase the stability of the left-to-right distribution of the modular robots over the initial shape and the final built shape. We then compare TBSR with a rival algorithm, C2SR (Cylindrical-Catoms Self-Reconfiguration), that can be used for the same scenarios. The objective of this work is twofold: proposing a new distributed algorithm based on an original idea of effort distribution. Secondly, provide another self-reconfiguration algorithm for 2D Catom system.

1 Introduction

We assume that the modular robots network is composed of a set of identical micro-robots communicating by physical contact and moving into a vertical 2D area. Each modular robot can be in contact with at most 6 regularly distributed modular robots. The modular robots describe therefore a 6-lattice grid as depicted in figure 10.

We assume also that a modular robot moves by rotating around one of its direct neighboring robot. The made move does not impact the position of any other robot. The robots communicate only with their direct neighbors by exchanging asynchronous messages. Therefore, each node

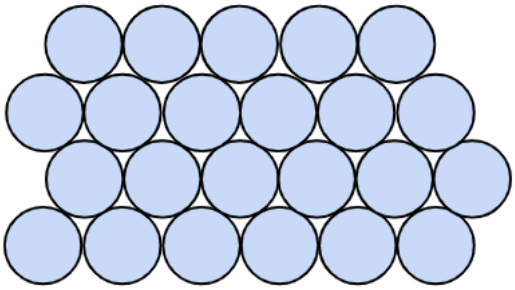


Figure 10: Modular robots are organized into 6-lattice grid where each modular robot has at most 6 direct neighbors.

has only local knowledge about its environment, but information such as the number of robots, current robots' shape, and relative coordinates are unknown initially. Besides, the modular robots have to prevent the splitting of the network into several separated components since the coordination of the reconnection could not be possible.

Recall that the main objective of a modular robots system is to provide a robotic machine able to change its properties in particular its shape to respond to some topological, operational, or user needs. Shape-shifting or self-reconfiguration problem refers to the algorithmic procedures allowing the set of modular robots to reorganize under a given target shape.

Almost all the works on the shape-shifting problem deal with specific cases of initial or final shapes [54, 63]. Some works attempt to address a wider range of shapes such as [71, 10] since they are rules-based oriented. In these latter works, the optimization of the reorganization cost according to the total number of moves which impacts the overall energy consumption, the maximum number of moves per robot, or convergence speed are ignored. In our work, we put more emphasis on the energy consumption.

2 Translation based self reconfiguration algorithm

In this section, we describe in details the Translation Based Self Reconfiguration algorithm (TBSR). First, we start by the space modeling presentation and the formulation of the target space. Then we explain the procedure of the TBSR algorithm.

2.1 Space modeling

The main feature of the Energy-aware shape-shifting algorithm is the data structure representing the final shape. First, the final shape is described without explicit reference to absolute coordinates for all the target positions. Instead, a more compact representation is employed to avoid managing an extensive volume of positional data. Therefore, the need for geographical positioning devices is obsolete. The final shape is described by two sets of diagonal; *on the floor* diagonals and *left-side* diagonals (see Figure 11).

This way of envisioning the final form is referred to as the *Diagonally-Layered Representation*(DLR). More formally,

Definition 2.1 (DLR Structure)**.** *We define DLR data structure as a set of two lists:*

$$DLR = \{LOTG, LOTL\}$$

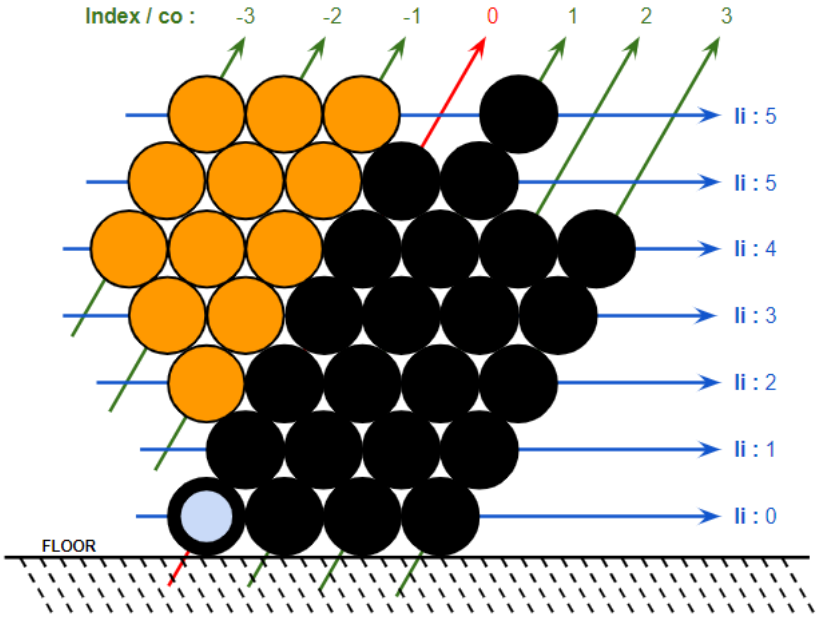


Figure 11: The final shape is composed of two parts: *on the ground* side (in black) and *the left side* (in orange).

Definition 2.2 (List On The Ground (LOTG))**.** *The "on the ground" side is described by an ordered list LOTG of diagonals from the left to the right:*

$$LOTG = \{(size_0, after_0), ..., (size_g, after_g)\}$$

Each element $(size_i, after_i)$ *indicates the size (number of robots diagonally stacked) of the* i^{th} *NE-SW diagonal and the number of all the nodes on the right side part of the final shape. The diagonal of index 0 is the westernmost diagonal starting from the ground line.*
It should be noted that:

$$after_i = \Sigma_{j=i+1}^{n} size_j$$

with $n = |LOTG|$.

For example, in Figure 11, $LOTG = \{(5, 14), (6, 8), (4, 4), (4, 0)\}$.

It is worth noting that the first line of the final shape (index 0) and the NE-SW diagonals of the final shape are continuous (i.e. there are no holes in the final shape after the end of the algorithm).

We observe that:

Proposition II.1.

$$after_i = after_{i+1} + size_{i+1}$$

Proof:

$$\begin{aligned} after_i &= \Sigma_{j=i+1}^{n} size_j \\ &= size_{i+1} + \Sigma_{j=i+2}^{n} size_j \\ &= size_{i+1} + after_{i+1} \end{aligned}$$

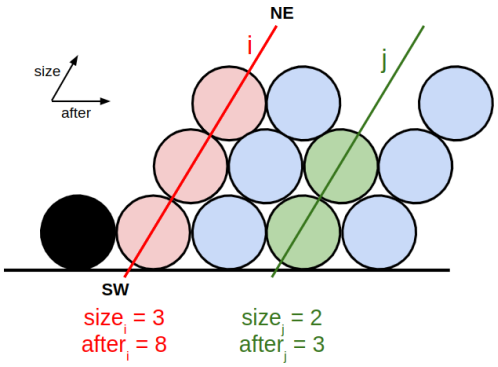


Figure 12: Example of values for $size_i$ and $after_i$. The black node is the bridge node.

with $n = |LOTG|$.

Definition 2.3 (List On The Left (LOTL))**.** *The left side of the final shape is described by an ordered list, LOTL, of diagonals from right to left this time:*

$$LOTL = \{(start_0, size_0), ..., (start_l, size_l)\}$$

An element $(start_i, size_i)$ gives respectively the line index (relatively to the ground) of the lowest position in the diagonal and the size of the diagonal in terms of number of positions.

For instance, in Figure 11, $LOTL = \{(1, 5), (2, 4), (3, 3)\}$.

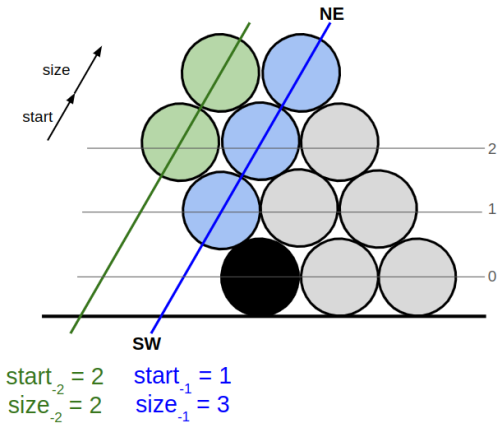


Figure 13: Example of values for $start_i$ and $size_i$. The black node is the bridge node.

2.2 TBSR Procedure

The Shape-shifting algorithm is composed of 4 consecutive steps. Some steps do not concern all the nodes.

2.2.1 Initialization step

During the initialization phase, each node determines its relative coordinates and then computes its evacuation order (as described in the next paragraph). We define the relative coordinates

LOTG		LOTL	
size	after	start	size
7	42	3	4
7	35	5	2
7	28		
7	21		
7	14		
7	7		
4	3		
2	1		
1	0		

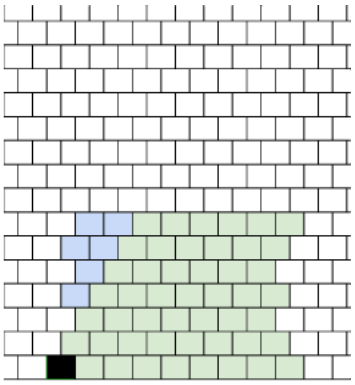


Figure 14: DLR for the Block shape of medium size.

of a robot as follows. Using a distributed algorithm, each robot determines its own line and column indices (li and co) relative to the lowest and leftmost nodes. The distributed algorithm for relative coordinates is given in Algorithm 1.

Algorithm 1: Distributed computation of the relative coordinates

local variables: l, c, Neighbor(X,Y): A Boolean function that returns true if there is a neighbor at the position X and Y, and false otherwise.
messages: sendToAllNeighbors(l, c): send l and c to all neighboring robots.
FromX(l, c): receive l and c from $X \in \{E, SE, SW, W, NW, NE.\}$
output: li,co
$li = 0$; $co = 0$;
if $\neg Neighbor(E, NE) \vee \neg Neighbor(SE, SW)$ **then**
  $sendToAllNeighbors(li, co)$;
when *the mesg* $FromEast(l, c) \vee FromNE(l, c)$ *is received* **do**
  if $c + 1 > co$ **then**
    $co = c + 1$;
    $sendToAllNeighbors(li, co)$;
end
when *the mesg* $FromSW(l, c) \vee FromSE(l, c)$ *is received* **do**
  if $l + 1 > li$ **then**
    $li = l + 1$;
    $sendToAllNeighbors(li, co)$;
end

2.2.2 Evacuation step

The evacuation order regulates and schedules the nodes' motions toward the final shape area. More precisely, a node, at the evacuation step, moves only if its priority is higher than all neighboring nodes. The move is done in the clockwise direction around the pivot node.

Rule 1. the freedom of movement rule. Because of possible mismatching issues due to physical constraints, a robot can only move from/into a cell if this cell is currently unoccupied and if no two symmetrically opposing cells adjacent to that cell are occupied (see Figure 16). Furthermore, we consider the floor as if it were filled with robots (We define *Floor* as the list of fictional robots that compose the floor). If a robot, r_i, satisfies the freedom of movement rule,

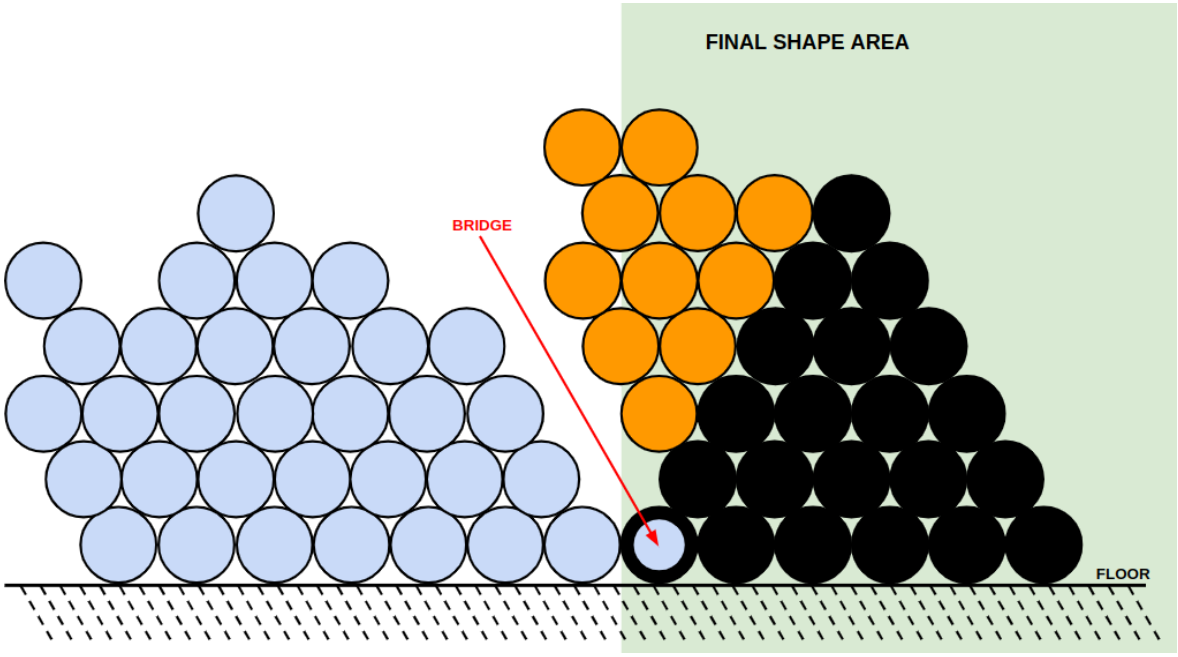


Figure 15: Shape shifting algorithm. The final shape area is delimited by the position at the East of the easternmost node on the ground of the initial shape. In blue the initial shape. In black and orange the final shape.

free(r_i) is true, otherwise it is false.

$$free(r_i) = \forall r_j \forall r_k, r_j, r_k \in \mathcal{N}_{p_i}^K \cup \mathcal{N}_{p_i'}^K \cup Floor, \{r_j; r_k\} \notin SymmetricNeighbors$$

$$SymmetricNeighbors = \{\{E; W\}; \{SE; NW\}; \{SW; NE\}\}$$

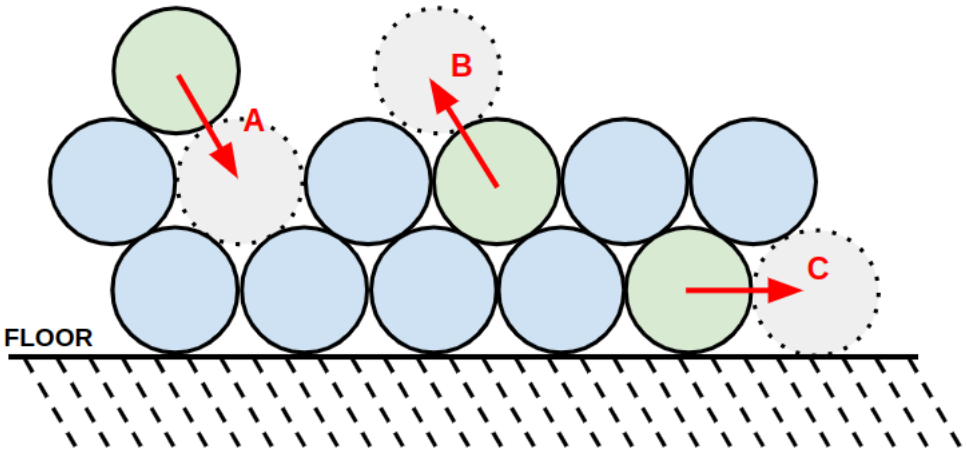


Figure 16: Examples of not allowed moves. (A) is not possible because of the two symmetrically opposing neighbors of the target position. (B) is impossible because of the two opposing neighbors of the initial position. (C) is impossible because the ground counts as a robot, meaning we have opposing neighbors.

Definition 2.4 (Order of a robot)**.** *We define the order of a robot r_i as follows:*

$$Order_i = 0 \quad \text{if on the ground}$$
$$Order_i = \text{co} + \text{li} \quad \text{otherwise}$$

The order of the nodes constituting the first line of the initial shape is fixed to the lowest value (0). Indeed, the ground line is evacuated at the end.

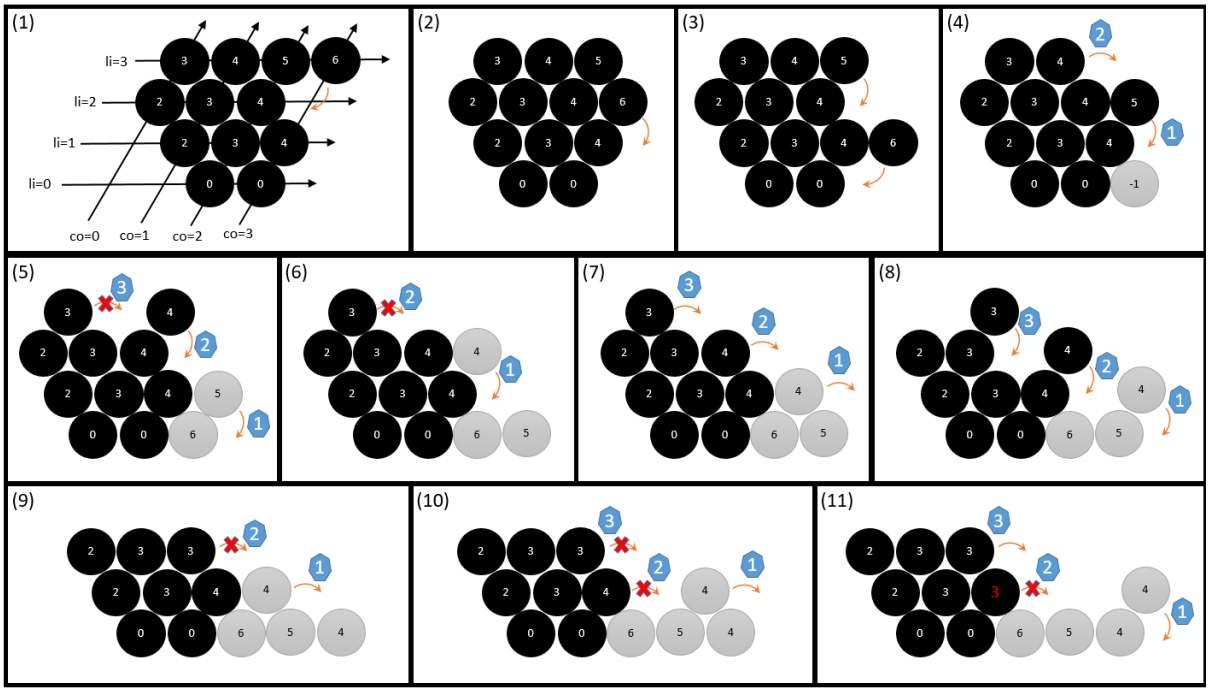


Figure 17: Example of initial shape evacuation. The nodes undergoing evacuation are depicted in gray and transition to the construction step. The arcs indicate the next motions and cross-marked arcs indicate blocked moves. The arc indexes refer to the order of the moves. The node index gives the order (definition 2.4).

Now, we formalize the priority relation between two nodes x and y in Definition 2.

Rule 2. movement triggering rule. Let us consider two robots r_i and r_j such that both r_i and r_j are on the periphery and r_j is the next peripheral neighbor of r_i in the direction of rotation, *d*. p'_{r_i} denotes the position that r_i would occupy after its rotation around r_j. r_i decides to roll towards its new position if the following logical condition is satisfied:

$$\begin{aligned} evacuation(r_i) = free(r_i) \wedge (\forall r_j \in \mathcal{N}_{p_i}^K, (r_i.order > r_j.order) \\ \vee (r_i.order = r_j.order \wedge \\ (r_i.SE() = y \vee r_i.E() = y \vee \\ r_i.SW() = y \vee r_i.W() = y))) \end{aligned}$$

The evacuation procedure allows the progressive erosion of the initial shape from the eastern-most NW-SE diagonals to the Western ones. To move around a pivot node, a node should have a higher priority than the pivot. To prevent blocking situations, if a node takes priority over its neighbors but it cannot move (physical constraints), it takes the same order as its NW neighbor. The NW neighbor receives priority over the blocked node and may unblock the evacuation

process. Figure 17 gives an illustration of the evacuation progress.

During the fifth step depicted in Figure 17, the move denoted as "3" is prohibited since the East neighbor of the move's pivot has a higher priority than the concerned node. However, the previously blocked move "3" at step (7) becomes unblocked whence the Est neighbor of the pivot moves (move "2" at step (7)). In step 10, both move "2" and move "3" become unfeasible; the move (2) is impossible since the node is surrounded by both NW and SE sides while the move (3) is impossible because the pivot is of higher priority. That is why the order of this latter is updated to the order of its NW neighbor at step (11). As a result, the move (3) becomes possible which unblocks the situation.

Rule 3. the collision avoidance rule. This rule requires local interactions with neighbors adjacent to its source and destination positions. A module r_i can move from its position p_i to the position p'_i if the following condition is satisfied:

$$progression(r_i) = evacuation(r_i) \wedge |\mathcal{N}^K_{p'_i}| \leq 3 \wedge (\nexists r_j \in \mathcal{N}^K_{p'_i} | r_i \neq r_j \wedge evacuation(r_j))$$

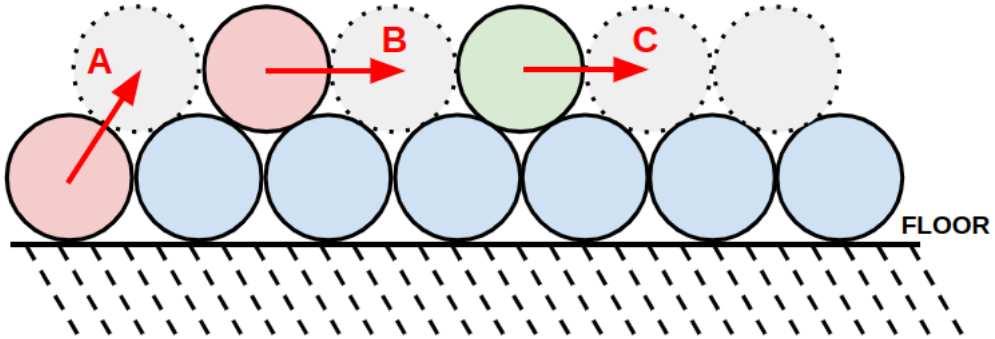


Figure 18: The movements (A) and (B) are prohibited by rule 3; (A) cannot move because (B) is active, and (B) cannot move because (C) is active. However, (C) is free to move as there are no obstacles in its path.

In Algorithm 2, we describe the evacuation process.

This evacuation step concerns the period during which the node leaves its initial position to reach the final shape area (see Figure 15). The evacuation procedure is executed in a manner that prioritizes the movement of eligible nodes. However, in order to avoid potential blocking scenarios, the evacuation sequence of a particular node may be modified to enable the evacuation of other nodes beforehand. Once a node successfully arrives at the final shape area, it proceeds to initiate the construction step.

2.2.3 Construction step

In the construction step, the nodes that have departed from the left side of the bridge node to the right side enter the construction state; they will take place on the ground line of the final shape or could be stacked according to the Diagonally-Layered Representation (DLR) scheme. In other words, during the construction step, the robot cross the ground line until one of these four cases occurs:

Algorithm 2: Evacuation algorithm

local variables: phase: state variable $\in$ {"*construction*","*evacuation*","*end*"}, initially "*evacuation*".
$diagIndex$, li, $nbpassages$: int, initially $diagIndex$ is 0 and $nbpassages$ is 0.
messages: $canMove(X)$: message sent to neighbor X to check if pivoting around it is allowed (i.e. $progression(self)$ is true). $X \in \{SW, SE, E, NE, NW, W\}$.
$HP(X)$: message sent to X to check if X has lower priority (i.e. $evacuate(self)$ is true).
local functions: $move(X)$: pivoting clockwise CW around X
output: moved away from initial position

```
while phase = "evacuation" do
    if HP(SW) ∧ canMove(SW) then
        move(SW);
        if li = 0 then
            phase = "end";
            diagIndex = 0;
    if HP(SE) ∧ canMove(SE) ∧ HP(SE.E) then
        move(SE);
    if HP(W) ∧ canMove(W) then
        move(W);
        if li = 0 then
            phase = "end";
            diagIndex = 0;
    if HP(SW) ∧ HP(SE) ∧ HP(W)∧ !canMove() then
        order = NW.order;
    if SE.phase = "end" ∧ SE.nbpassages = after_diagIndex + size_diagIndex − 1 then
        phase = "leftSideConstruction";
    if SE.phase = "end" ∧ SE.nbpassages < after_diagIndex + size_diagIndex − 1 then
        phase = "construction";
    if SW.phase == "end" then
        phase = "construction";
        SW.nbpassages + +;
end
```

1. the node reaches the end of the constructed ground and the ground is not complete yet: the node joins the ground line. The crossed nodes already on the final ground count the number of passages. When the number of passages reaches the needed number of nodes to build the right side of the final shape, the node forbids further passages.

2. the node reaches the end of the constructed ground and the ground is complete: The node is then the second node of the easternmost diagonal.

3. the node during the final ground crossing meets a ground node that does not allow the passage: The node deduces that it is the second node of the current diagonal.

4. the node meets the second node of the next diagonal: According to whether the met diagonal is complete or not, the node becomes the second node of the current diagonal or starts climbing the joined diagonal. In this latter case, the node climbs the met diagonal until it reaches its top. The node joins the diagonal.

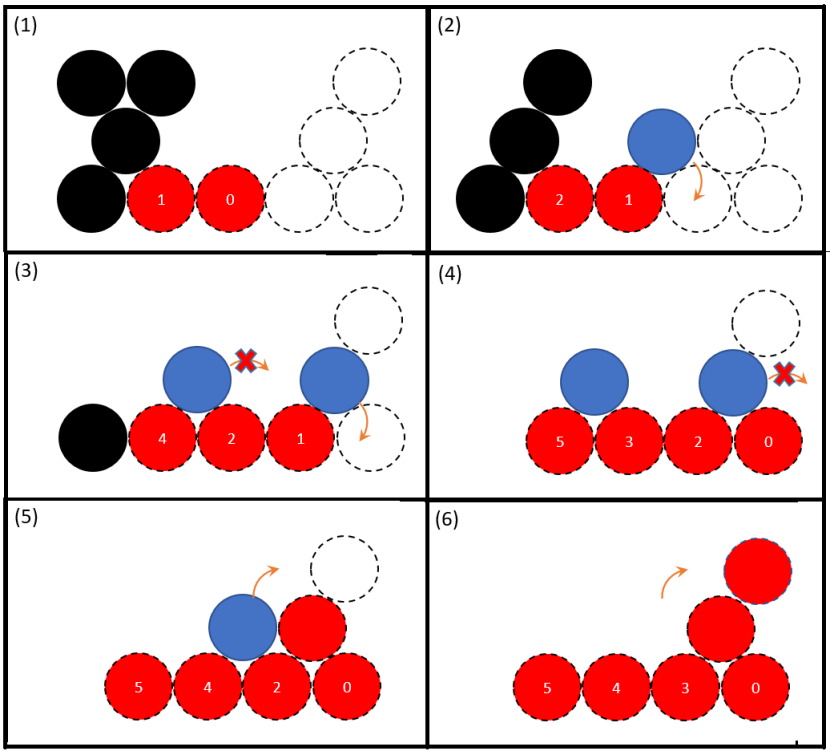


Figure 19: Example of construction of 6 nodes final shape $LOTG = \{(1,5);(1,4);(3,1);(1,0)\}$. We indicate in black the nodes at the evacuation phase, in red the nodes at the end phase, in blue the nodes at the construction phase, and in white the empty final positions. The numbers in the node designate the number of passages *nbpassages* counted by the already built ground nodes.

Three of the cases described above are illustrated in figure 19. When a node reaches its final position, the phase state is set to ”end”. In the sub-figure (2), the node under construction phase (in blue) moves around the SW neighbor to join the ground line (1st case). In the sub-figure (3), the right blue node behaves as described in the first case and joins the ground line, while the second blue node avoids moving on to eventually not block the previous node (according to the rule 3). At step (4), the right node of the second line cannot move to the right because the SE

neighbor does not authorize the passage. The node deduces that it reaches a final position (case 3). To determine that the right side of the final shape is built or that enough nodes crossed to the right side, a ground node at diagonal d uses the following rule:

Rule 4. the forbidden passage rule. This rule is used by ground nodes to determine if the right side of the final shape is built :
This rule is used by nodes in the ground line of the goal shape to determine whether they can be used as a pivot by a neighboring node if *nbpassage* is *true* :

$$nbpassages < after_d + size_d - 1$$

Where d is the diagonal index of the concerned node ($d = diagIndex$ in Algorithm 17).

These cases are illustrated in Figure 19. For instance, in sub-figure (4), the right blue node, asks the ground node in the last diagonal (of index 3) if the passage is allowed. The rightmost node at the moment has counted 0 passages so far and the values of $size_i$ and $after_i$ for its diagonal (index 3) are $size_3 = 1$ and $after_3 = 0$. Since, we have $after_3 + size_3 - 1 = 0$ and $nbpassage(0) \not< after_3 + size_3 - 1(0)$, the rule is violated, therefore the ground node forbids the move to its diagonal.

In sub-figure (5), the last blue node (in the construction phase) cannot move to the East position because it is occupied by the second node of the penultimate diagonal. Since the SE neighbor of the node allows the passage, the node climbs the penultimate diagonal (index=2) until it reaches its top. The construction procedure is given in Algorithm 3.

2.2.4 Left side construction

Once a node reaches the bridge node and the bridge node has already reached its maximum number of passages (*nbpassage*), the node transitions from the *evacuation* state to the *left_side_construction* state. During this step, the node moves along the lower side of the constructed shape until it reaches a new diagonal or completes the last incomplete diagonal.
The left-side construction procedure is depicted in figure 20. The left side is composed of 3 positions represented by the white circles and described by a $LOTLS = \{(2,1);(1,2)\}$. The nodes with the *left_side_construction* state (in green) run along the SW side of the constructed shape (in red). A node checks if the East diagonal is completed and if it is not the case, the node climbs (see Figure 20.(3)) the East diagonal until it reaches the top and joins it (see Figure 20.(5)). If the East diagonal is completed, the node search of the start position of the new diagonal (see 20.(6)).

3 Experimental results

In this section, we study the performances of the Translation-based Self Reconfiguration algorithm in terms of the total number of moves, the maximum number of moves per robot, and the standard deviation of the number of moves over the robots, denoted respectively by Sum, Dev, and Max. When in a self-reconfiguration with n robots, where a robot i end up doing $nbMoves(i)$ movements, they are defined as :

$$Sum = \Sigma_{i=1}^{n} nbMoves(i) \tag{1}$$

$$Max = max_{i=1}^{n} nbMoves(i) \tag{2}$$

Algorithm 3: Construction algorithm

local variables: phase: state variable $\in \{$"$construction$","$left\_side\_construction$","end"$\}$, initially "$construction$".
$diagIndex$, li, $nbpassages$: int, initially $diagIndex$ is 0 and $nbpassages$ is 0.
messages: $canMove(X)$: message sent to neighbor X to check if pivoting around it is allowed (i.e. $progression(self)$ is true). $X \in \{SW, SE, E, NE, NW, W\}$.
$HP(X)$: message sent to X to check if X has lower priority (i.e. $evacuate(self)$ is true).
local functions: $move(X)$: pivoting clockwise CW around X
output: reach final position in goal shape.

```
while phase = "construction" do
    if SE = NULL ∧ canMove(SW) then
        move(SW);
        diagIndex = W.diagIndex + 1;
        phase = "end";
    if
     SE.nbpassages < after_diagIndex + size_diagIndex − 1 ∧ canMove(SE) ∧ (SE.E.NE =
     NULL ∨ SE.E.NE.phase = "end") then
        move(SE);
        diagIndex = SW.diagIndex;
        SE.nbpassages + +;
    if SE.nbpassages < after_diagIndex + size_diagIndex − 1 ∧ E! = NULL ∧ E.phase =
     "end" ∧ canMove(E) then
        move(E);
        phase = "left_side_construction";
    if SE.nbpassages = after_diagIndex + size_d − 1 then
        phase = "end";
end
while phase = "left_side_construction" do
    if E! = NULL ∧ E.phase = "end" ∧ canMove(E) then
        move(E);
    if E = NULL ∧ canMove(SE) then
        move(SE);
        phase = "end";
end
```

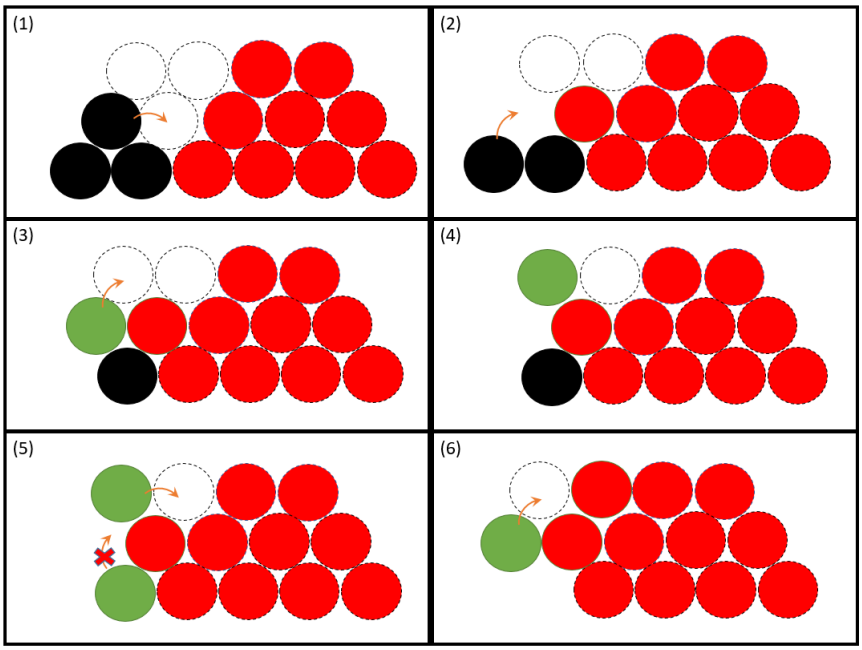


Figure 20: Construction of the left side. White circles represent the target positions, red ones are the already fixed nodes, and green ones are the nodes at the *left side construction* state.

$$Dev = \sqrt{\frac{\Sigma_{i=1}^{n}(nbMoves(i) - \frac{Sum}{n})^2}{n}} \tag{3}$$

These three criteria allow appreciation of the quality of the trade-off between the overall amount of energy consumed by the robots and how this effort is spread over the set of the robots. We compare TBSR algorithm to C2SR algorithm.

It is worth noting that that the two algorithms differ mainly in the deconstruction and construction order of the initial and final shapes respectively. Whereas C2SR prioritizes the deconstruction of the top horizontal lines of the initial shape to construct the bottom lines of the final shape, TBSR prioritizes the deconstruction of the right side diagonals of the initial shape to construct the right side diagonals of the final shape first.

The two algorithms are compared using VisibleSim simulator [79]. For the relevance of the study, we considered several scenarios representing different classes of shapes (12 classes) with different mass distributions (symmetric, asymmetric, flattened, high, etc) and 3 different sizes. This results in 432 scenarios generated by the script we developed VisibleDraw (see 4).

Figure 21 compares the Sum and Max of C2SR and TBSR according to the final shape whatever is the initial shape (average over 12 initial shapes). The results show that TBSR outperforms C2SR in terms of both criteria.

3.1 Effect of shape size

In Table 3, we summarize the results obtained for the 432 different scenarios. For each category of size, we give the overage over the 144 scenarios of the sum of required moves, the maximum number of moves per robot, and the standard deviation of the number of moves over robots. First of all, Table 3 shows that TBSR outperforms C2SR according to the three selected criteria.

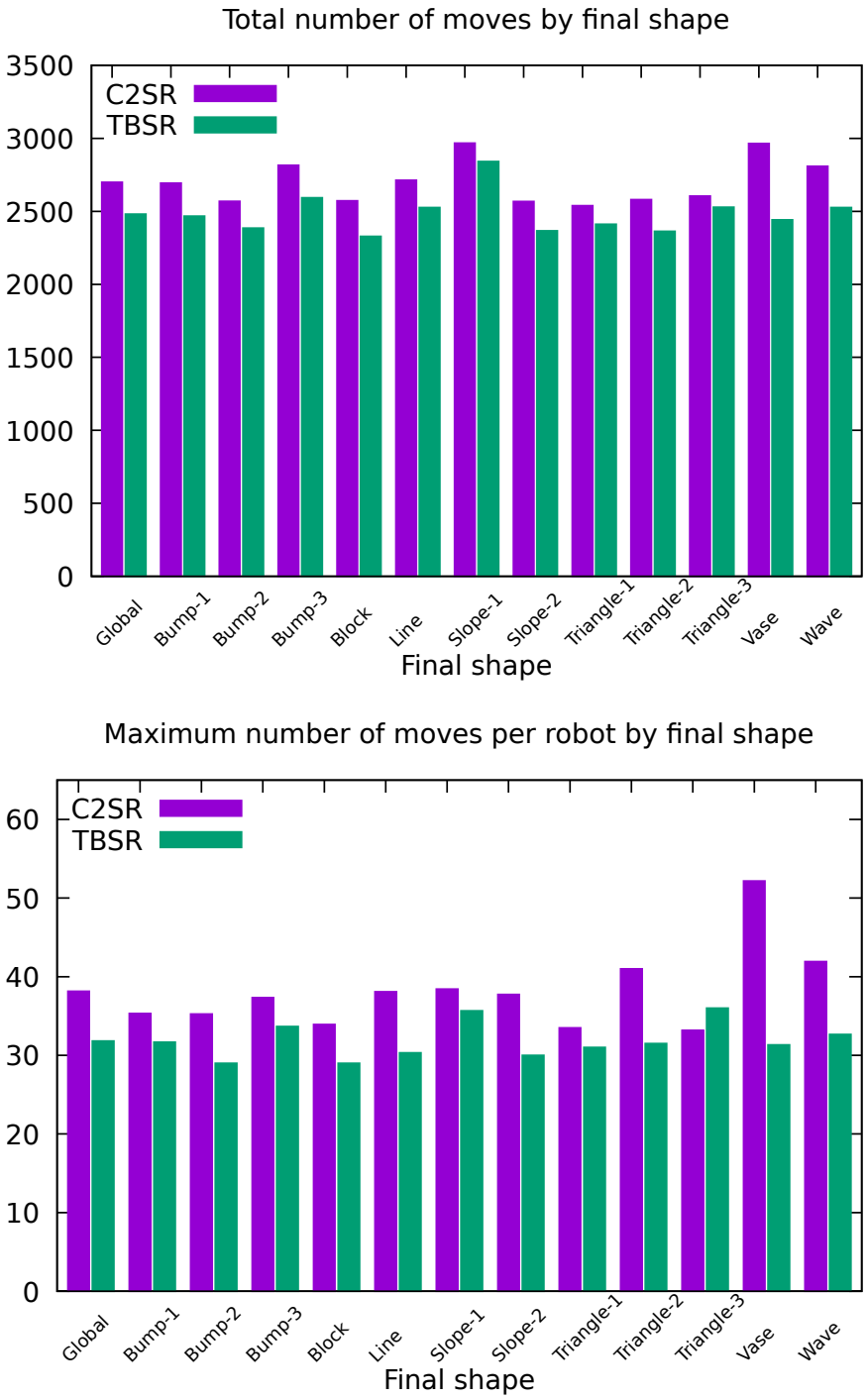


Figure 21: Comparison of *Sum* and *Max* obtained by the C2SR and TBSR algorithms on the 144 large scenarios. Each value represents an average of 12 initial shapes to reach a given final shape.

	C2SR			TBSR		
	Sum	Max	Dev	Sum	Max	Dev
Small	66,5	10,1	3,0	63,9	10,3	3,2
Mid	860,5	25,4	5,3	840,0	24,3	5,2
Large	2699,8	38,2	7,2	2481,5	31,8	5,6

Table 3: Average sum of moves (Sum), maximum of moves per robot (Max), and standard deviation of the number of moves (Dev) over the robots according to the size of the scenario. Each line corresponds to an average of over 144 scenarios.

The benefit of translation-based reconfiguration is more significant when the size of the network of modular robots is higher. The average number of moves required to reach the final shape, for large shapes, decreased from 2699 moves to 2481 moves, which corresponds to a gain of 8%. At the same time, the maximum number of moves done by a robot decreased from 38 to 31, which proves that TBSR provides a better distribution of the energetic effort over the modular robots than C2SR. This observation is confirmed by the standard deviation which is lower in TBSR.
Figure 22 shows the evolution of the three criteria Sum, Max, and Dev according to the scenario size (10, 55, and 120 robots respectfully for Small, Medium, and Large). In this scenario, the initial shape represents a triangle while the target shape is a wave. The three graphics show that the TBSR approach improves slightly the total number of moves. However, the most significant improvement is observed in terms of the Max and Dev criteria, which reinforces the intuition that a mass translation-based approach leads to a better effort balancing over the modular robots.

3.2 Effect shape asymmetry

To study the impact of the horizontal mass distribution of the initial and final shapes on the performances of TBSR algorithm, we consider the *slope* shapes shown in figure 23. In the first case (on the top), the final shape represents a simple translation of the initial shape (slope-1). In the second case, the final shape (slope-2) corresponds to the exact symmetric shape of the initial shape (slope-1). We can see that the closest modules from the bridge module in the initial shape are the farthest in the goal shape. In contrast, the farthest modules in the initial shape are now the closest to the bridge in the final shape. This characteristic of TBSR ensures that each node (which is not part of the initial or final ground line) moves an approximately equal distance on average.
The numerical results obtained by C2SR and TBSR on these two cases are given in Table 4. As expected, Slope-1 to Slope-1 case needs more moves to reach the final shape. Indeed, in this first case, the large majority of the robots at the final state are at right far from the initial shape.

In both cases, TBSR outperforms C2SR as it requires fewer moves. Moreover, we observe that TBSR outperforms more significantly C2SR in the second case. The reason is illustrated in Figure 23. In this figure, we use different colors to show the distribution of the modular robots in the initial and final shapes. In the slope1 to slope-1 case, we show that some robots in the orange area cross a large distance before reaching their final positions. This situation disappears in the slope-1 to slope-2 case, where we observe that after the evacuation of the orange area in the initial shape, the robots do not climb a long diagonal before reaching the final positions.

The performances of TBSR are more appreciated (see Figure 21) when the top side of the final shape is concave. Shapes such as vase or wave (see Figure 24) present a weak point of C2SR.

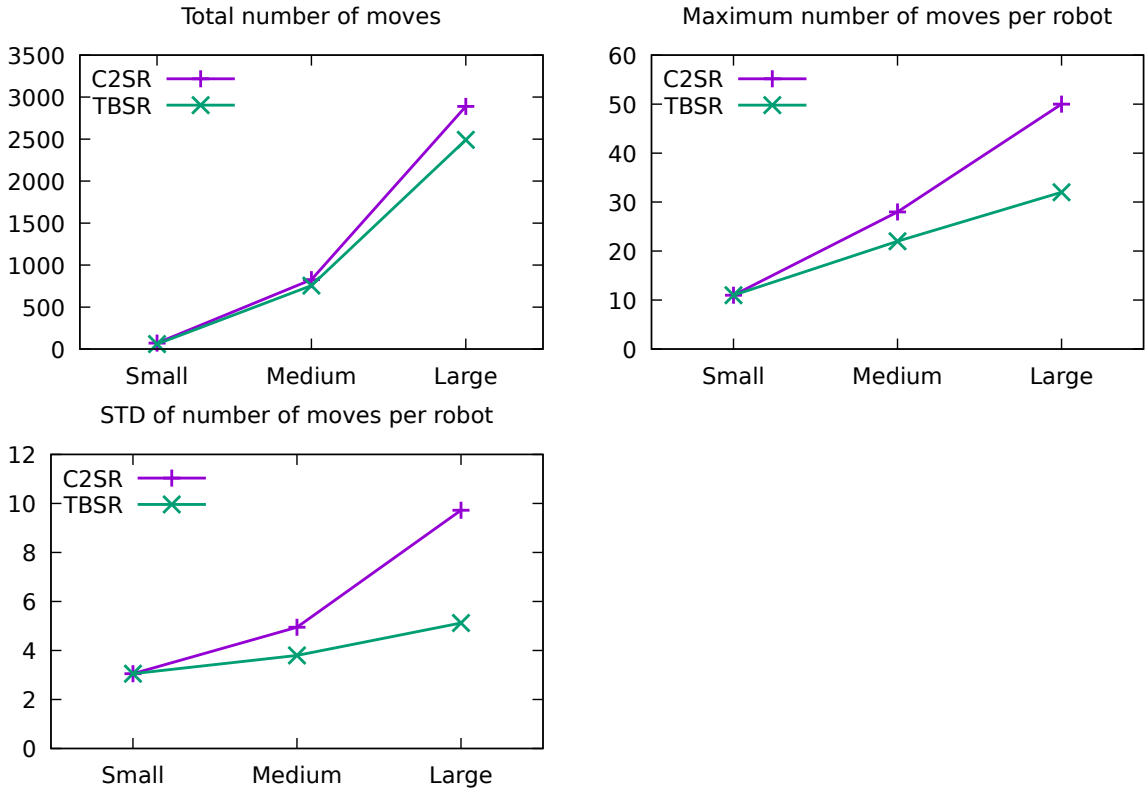


Figure 22: Evolution of *Sum*, *Max*, and *Dev* criteria according to the size of the network of modular robots. The initial shape represents a triangle while the final shape represents a wave. Small is with 10 robots, Medium is with 55 robots and Large is with 120 robots.

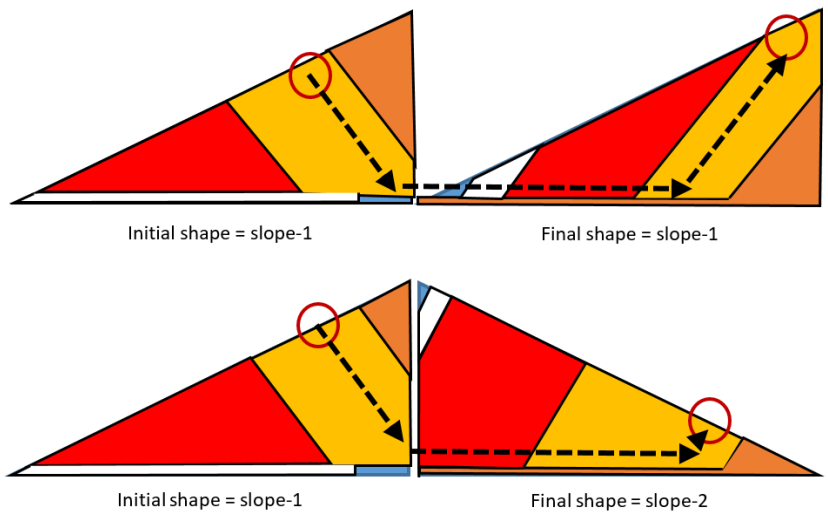


Figure 23: Modular robots distribution maps in initial and final shape using TBSR algorithm. The final shape is constructed in the following order: brown, orange, red, white then blue part. At the top, the self-reconfiguration of slope-1 shape to the same shape. At the bottom, the self-reconfiguration of slope-1 shape to slope-2 shape.

The performance of TBSR is particularly notable (see Figure 21) when the top side of the final shape exhibits a concave shape. Shapes like a vase or wave (see Figure 24) reveal a weakness

	C2SR			TBSR		
Final shape	Sum	Max	Dev	Sum	Max	Dev
Slope-1	2778	40	6,69	2742	34	6,66
Slope-2	2382	36	5,99	2267	29	4,33

Table 4: Comparison of C2SR and TBSR on two scenarios: slope-1 to slope-1 reorganization, and slope-1 to slope-2 reorganization

in the C2SR algorithm due to its horizontal layering strategy, in contrast to our approach that proceeds diagonally.

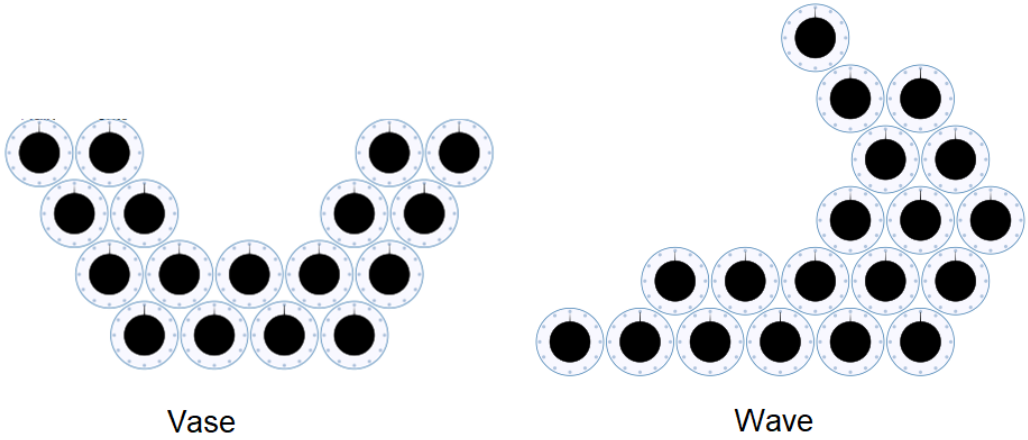


Figure 24: Wave and vase shapes

3.3 Energy consumption cartography

Figure 25 shows the energy consumption map for the two re-configurations scenarios: Slope-1 to Slope-1 and Slope-1 to Slope-2. We observe that the distribution of the robots that performed many more moves (in light) is different. This is a direct effect of the shapes' asymmetry. The set of robots surrounded in red corresponds to the area discussed in Figure 23 with the case Slope-1 to Slope-1. This set of three robots is lighter than their left and right neighbors. This means that the energy consumption map is not just a color scale diagonal from light to dark. The same phenomenon is observed in the second case (Slope-1 to Slope-2). Figure 25, at the bottom, shows two robots with a number of moves higher than neighboring robots.

4 Conclusion

Self-reconfiguration for modular robots systems is a challenging problem in distributed programming. Most works in this field deal with specific cases representing a subset of initial or final shapes. Additionally, few studies have addressed the optimization of energy performance in the reconfiguration process.

To assess the performance of TBSR algorithm, we compare it to C2SR algorithm over 432 scenarios with different sizes and shapes. The obtained results show clearly that TBSR leads to a better reconfiguration process with less required moves (up to 17% reduction), and a better energy effort balancing over the robots (up to 40% Max reduction).

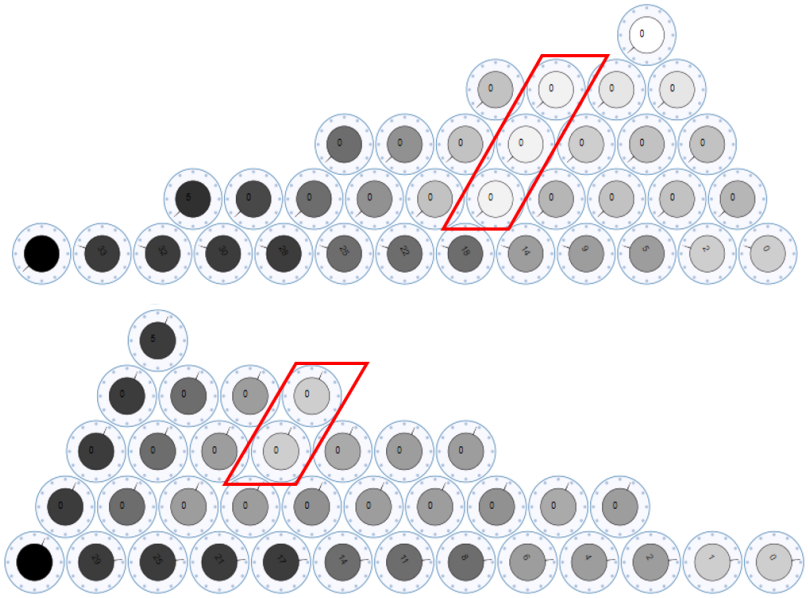


Figure 25: Energy consumption cartography when the final shape is reached by TBSR algorithm: At the top, when the final shape is Slope-1 and at the bottom when the final shape is Slope-2. The initial shape for both cases is Slope-1. The grey level indicates the energy consumption, darker robots are the less moved robots.

The comparison between TBSR and C2SR demonstrates that while TBSR generally outperforms C2SR in terms of the total number of moves metric, C2SR still be more efficient in certain cases. Therefore, studying how to pre-process problem inputs to determine which method is better suited for a given problem is an interesting approach. This approach allows for a hybridization of different self-reconfiguration methods, leveraging the advantages of existing methods. AI methods offer a promising approach to achieving such preprocessing. AI can be used to classify a self-reconfiguration instance according to the expected best self-reconfiguration method to use.

Chapter III
Artificial intelligence-based self-reconfiguration approach

1 Introduction

Now that we proposed a new self-reconfiguration algorithm that presents some advantages compared to C2SR algorithm, how to prevent using TBSR when C2SR is better and vice-versa? In this chapter, we describe an original and hybrid approach based on artificial intelligence for the selection of the best self-reconfiguration algorithm for a given instance, called (CNNSR). The objective is to introduce a new mechanism that allows programmable matter to behave smarter according to faced situation. Even if the presented work considers only TBSR and C2SR algorithms, it is obvious that the described procedure can be extended to introduce other distributed algorithms.

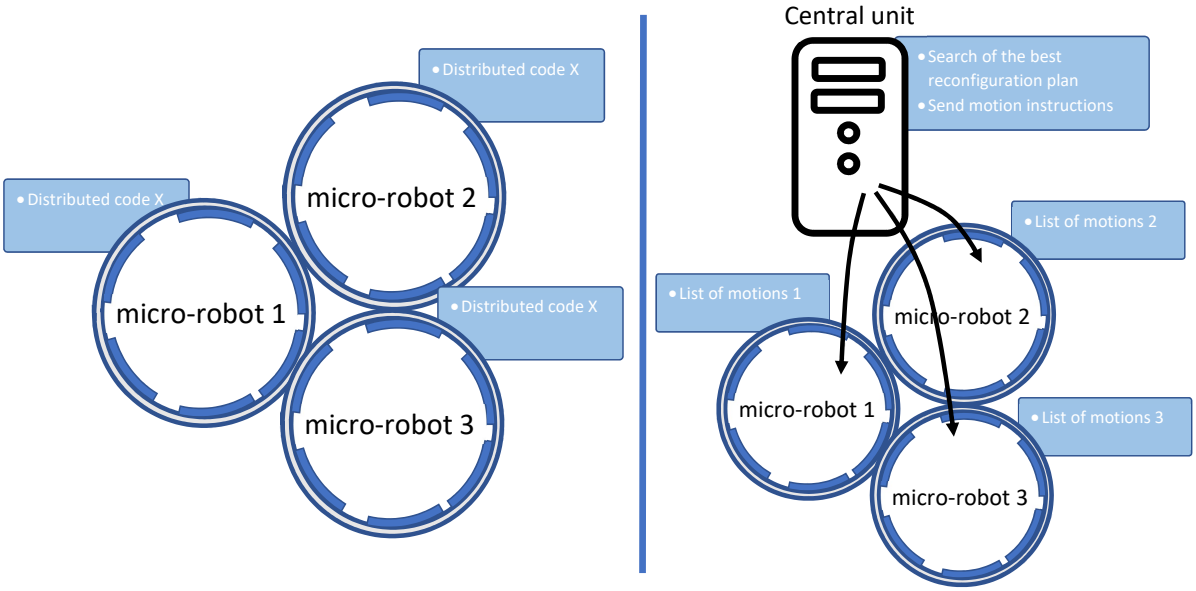


Figure 26: The working scheme of MRSR algorithms: In centralized approaches (right side), robots simply execute their own list of motions sent by the central unit, while in distributed approaches (left side), robots execute the same local rules based code to determine their motions.

2 Contributions

As shown in table 5 underlines the contribution of our approach in contrast to both centralized and distributed approaches across 8 key attributes: robot's memory requirement, centralized external memory requirement, the motion plan pre-computation time, the reconfiguration time to the final shape, robots' energy consumption, fault-tolerance, and communication requirement (inter-robots and robots-central unit). The table shows that centralized and distributed approaches each offer advantages and drawbacks. In this context, our hybrid approach using a CNN-based preprocessing phase, mitigates the major drawbacks of each approach and provides, therefore, a good compromise. For instance, centralized approaches necessitate a significant pre-

processing time before the start of the self-reconfiguration process due to the time requirement to compute the best reconfiguration plan. On the contrary, distributed approaches eliminate the preprocessing stage, but escalate the energy, memory, and communication complexity due to real-time motion computation. Our hybrid approach integrates a succinct online preprocessing phase (CNN) which in turn facilitates the selection of a fitting distributed algorithm to be executed across the robots during the subsequent reconfiguration phase. The designated algorithm is expected to reduce the costs to the robots and improve the overall performance of the reconfiguration process. The process of training the Convolutional Neural Network (CNN) necessitates an intensive offline processing phase. However, it is noteworthy that this training phase can be done once as a precursor, before being used as many times as required.

Table 5: Strengths and weaknesses of centralized (centr.) and distributed (distr.) approaches compared to our hybrid approach (CNNSR). High means the approach is very efficient for the characteristic, and low means it is not.

	center.	dist.	CNNSR
memory storage (robots)	low	average.	avg.
memory storage (external)	high	none	low
planing time	high	none	low
reconfiguration time	optimal	avg.	sub-opt.
energy consumption (robots)	low	avg.	sub-opt.
robustness	none	may be	may be
comm. (inter-robots)	low	high	avg.
comm. (central unit-robots)	high	none	low

This chapter is organized as follows: Section 3 gives an overview of the MRSR methods proposed in the literature and then highlights the interest of the proposed hybrid approach. In section 4.1, we discuss the global working scheme of our Artificial Neural Network-based Modular Robots Self Reconfiguration. Section 5 describes the different components of the implemented Convolutional Neural Networks (CNN). Section 6 details the CNN training process, the used data-set, and the obtained results. A conclusion and a presentation of the main perspectives of this work are given in Section 7.

3 Related works

Modular reconfigurable Robots (MRR) represent a robotic system where a set of tiny mobile modules are able to reorganize themselves into a given shape or topology. Many modular robots platforms were proposed such as: ATRON [42], MTRAN3 [43], Polybot [103], Roombot [8], Smores [41], M-Blocks [84], RoboGen [7], SmartBlocks [82], 2D-Catom [1], Datoms [80], Kilobot [85]. In [105], Li Zhu identified more than 121 MRR platforms from 1988 to 2018. Mainly these solutions could be classified according to four questions:

- do the modular robots require physical contact to be able to communicate and/or move ?
- do the modular robots distribution describe a lattice grid?
- do modular robots movements impact on the positions of the other robots?
- do modular robots movements change their neighbor relationship with the other robots?

.

Regardless of the targeted MRR platform, the algorithmic solution allowing the modular robots to reach a final shape, problem called Modular Robots Self-Reconfiguration (MRSR), represents a major challenge. MRSR approaches belongs to two major categories: centralized, or distributed.

3.1 Centralized approaches

The centralized approach has the potential to provide more efficient solutions, i.e. a better number of moves and a faster convergence time. In addition, robots work as executive units which reduces their computational and memory requirements. However, centralized approaches are less fault-tolerant: first, the central unit makes a single point of failure of the whole system; second, should some robots encounter errors while executing their instructions, the global convergence could fail and require computing a new solution. Third, the transmission of motion instructions to modular robots remains a real challenge. Fourth, centralized algorithms do not scale well, while programmable matter will easily reach tens or hundreds of thousands of nodes. In [21], we see that the computation time for the sequence of movements is exponential.

Despite these drawbacks, many works follow this working scheme [31][16][55]. In [16], the authors proposed to use a similarity metric between the target shape and the intermediate shape to guide future actions in order to increase this similarity. The similarity metric estimates the distance between the goal positions and the current positions. In [55], the similarity metric is computed according to the Euler tour surrounding the current and target shapes. The main drawback of such approaches is the relevance of similarity metrics. Indeed, a simple computation of the similarity between an intermediate shape and the final shape neglects the robots' motion constraints. In other words, the shift from one shape to another similar shape could be impossible due to collision and friction constraints. In [62], authors modeled the MRSR problem as a linear programming problem. However, the model ignores the connectivity constraint (robots should remain connected all together) of the robots, which makes the provided solutions not directly operable. In [31], authors modeled the MRSR problem as a SATisfiability problem (SAT), but the connectivity and frictions constraints were ignored.

3.2 Distributed approaches

In distributed self-reconfiguration algorithms, moves are executed as soon as they are planned, while in centralized self-reconfiguration algorithms, moves planning is an offline phase followed by an online moves execution phase. This property allows the distributed approach to be more flexible since the algorithm can react and adapt to failures. In [60], authors proposed a distributed algorithm for 4-lattice robots evolving in 2D horizontal space (Catom). The algorithm builds the final shape by decomposing it into multi-layer curves. When the failure of a micro-robot is detected, the neighboring robots trigger an alternative actions to prevent the blockage. However the solution suffers of high time convergence due to a low coordination level between robots. In [58], authors proposed a particular distributed approach for SMORES robots. Indeed, the algorithm starts by selecting a root module, which computes the reconfiguration plan. The approach assumes that each micro-robot is able to manage the computation of the global plan. Therefore, even if a dedicated central unit is missing, the authors assume that the root robot plays this role. In [11], the authors modeled the problem by a discrete trajectory optimization problem. The robotic modules compute, based on minimum hop count strategy, the shortest path between two points of the modular surface. In [10]. the self-reconfiguration algorithm starts with a centralized phase where contiguous substrate path is computed that represents a subset of the target positions. Once the substrate computed, the robots move to

fill the substrate path and then the positions at the north and the south of the substrate path. The algorithm assumes that the initial shape is a linear chain which limits its usage.

3.3 Analysis of related work

Self-reconfiguration algorithms usually target a specific type of robot with deeply different properties (see for instance [86, 75, 13]). Consequently, extending the use of a an algorithm to other hardware modular robots platforms is not easy task, making the comparison of different algorithms difficult. On top of that, even when comparing algorithms on a unique type of robot, which can be achieved by a simulator [97], global results (success, moves count, communications count) are easy to extract and compare, while these global metrics root causes are a lot less clear. Among the properties of self-reconfiguration problem input parameters, one can think of: the number of robots, initial and final shapes' outer envelopes, convexity/concavity, compactness/distortion, the distance between initial and final shape, the overlapping area between initial and final shapes, etc. In terms of solutions KPI, one can think of many among whose: total number of moves, maximum number of moves per robot, standard deviation of the number of moves over the robots, number of exchanged messages, convergence time, etc.

In this context, supervised classification using machine learning methods [92] can help identify the most suitable self-reconfiguration algorithm according to the problem parameters.

As described in Part II, the final positions (target shape) are expressed as a set of points known by their coordinates (x,y), where x designates the diagonal index and y the line index relative to the leftmost diagonal and lowest line as illustrated in figure 27.

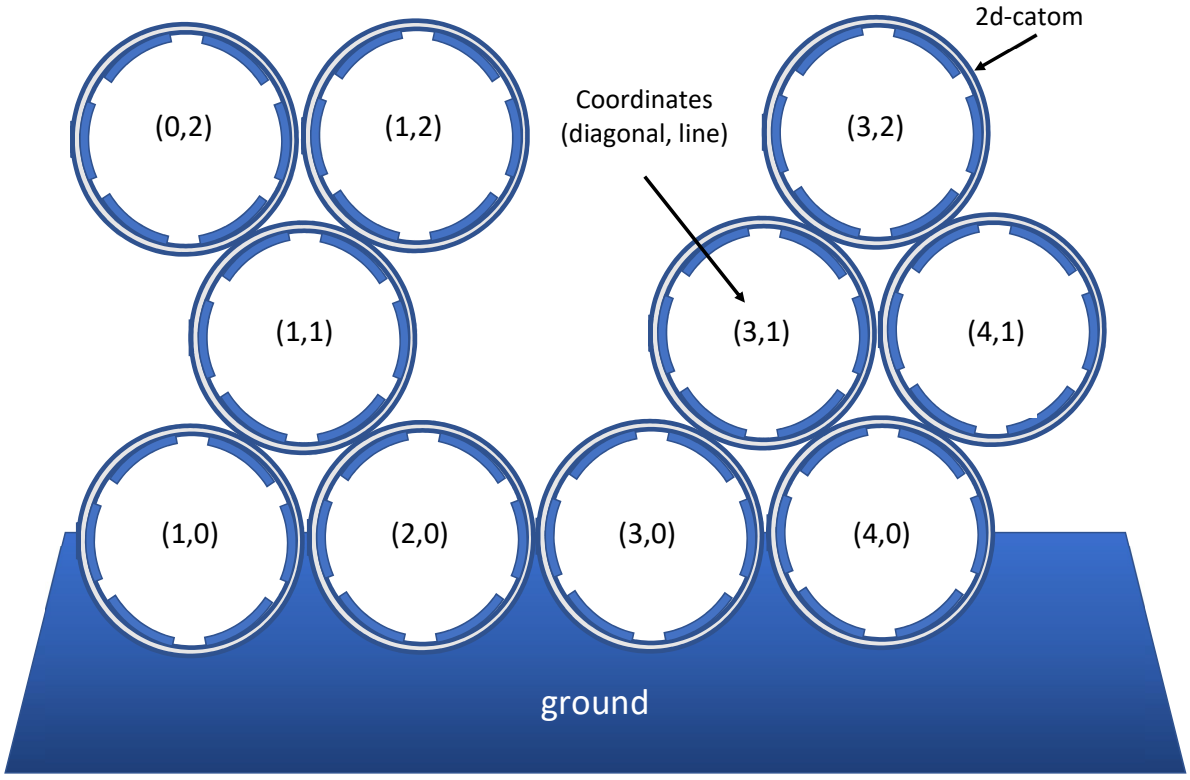


Figure 27: 2d-catom modular robots organization: the micro-robots are placed on a horizontal plane (ground) and are stacked following 6-lattice scheme.

4 CNN based Modular Robots Self Reconfiguration

Our approach called CNNSR is a hybrid centralized/distributed approach. The algorithm carries out two steps as depicted in Figure 28. In the first step, an Artificial Neural Network identifies the suitable distributed reconfiguration algorithm for a given scenario on the basis of the problem characteristics. Then the selected distributed algorithm is run on all the micro-robots.

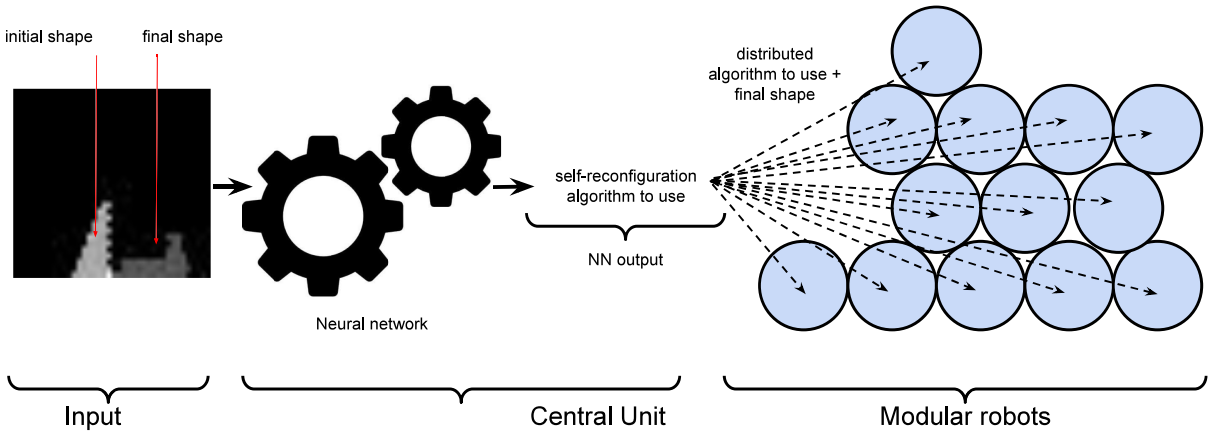


Figure 28: Working scheme of our hybrid approach: first the Neural Network, select the more suitable distributed self reconfiguration algorithm according to the initial and final shapes. Then the micro-robots are informed of the selected algorithm to run and the final shape to reach.

4.1 Selection of the suitable distributed algorithm step

The ANN operates as a classification method of the MRSR scenarios. Indeed, the Modular Robots Self Reconfiguration problem presents 3 of the characteristics [83] that promote ANN use:

- *imbalanced data*: the performance analysis of the studied self-reconfiguration methods may show that some methods generally outperform other methods. Therefore, the input data may present a few cases in favor of some methods.
- *incomplete data*: The diversity of modular robots' self-reconfiguration problems makes difficult the correct sampling of all possible and relevant cases. Therefore the input data are potentially partial.
- *high-dimensionality*: MRSR problem presents a massive number of characteristics, which promotes the use of ANN to discover the latent factors that define the nature of a given scenario.

More formally, let P be the set of available, either pre-loaded or ready to be deployed on the robots, distributed algorithms on the micro-robots. A MRSR problem is defined by a matrix M as an 8-bits gray-scale image that describes the planar 6-lattice grid in terms of robots locations in the source and target shapes (see fig. 29 for an example). The lines of the matrix designates the positions with the same coordinates y, while the columns of M designates positions with the same diagonal index x. The value of $M(x, y)$ is equal to 0 if the position is neither belonging to the initial shape nor the target one. $M(x, y)$=82 (resp. 170) means that position (x, y) only belongs to the initial (resp. final shape). Finally, $M(x, y)$=255 if the position both belongs to the initial and final shapes. The CNN module determines according to the Matrix M, the most

suitable distributed algorithm $p \in P$ to reach the final shape.

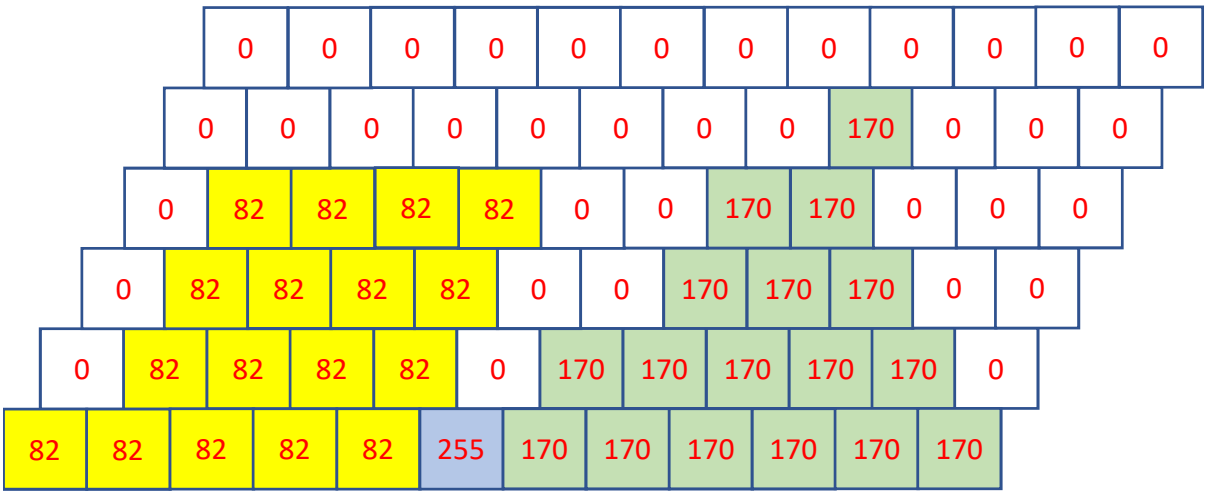


Figure 29: Example of MRSR problem coding: The initial and final shapes are coded by a 2D matrix M. Each cell corresponds to a position in the 2D vertical plane where robots move. A value 82 in the cell means that the position is occupied initially by a robot, 170 means that the position should be occupied at the end, and 255 means that the position is occupied in both initial and final shapes.

MRSR problem is a multi-criteria optimization problem. In this chapter, the suitability of a given distributed algorithm is considered according to different metrics: total number of moves over the robots, maximum number of moves per robot, and standard deviation of the number of moves over robots.

Artificial Neural Networks (ANN) exist since the beginning of computer science [100, 39]. Artificial Neural Networks have been widely used in many fields [90, 89, 52], including robotics [91, 38, 98] such as image processing, facial recognition, voice recognition, and computational learning . These works give birth to different categories of ANN:

- Multilayer perceptron (MLP) Neural Networks [73] are designated for processing vector input to analyze the data features. This kind of ANN has been used for text classification and simple image recognition.
- Convolutional neural network (CNN) [37, 26] also called ConvNet presents a deep feed-forward architecture. It is widely used for image and video processing due to its ability to identify the spatial relation between pixels.
- Recurrent Neural Network (RNN) [87, 23] is a class of ANN dedicated to the analysis of time-series data and other sequential data like speech recognition or processing.
- Graph Neural Network (GNN) [101, 36] is adapted to the analysis of graph data that relate the direct and indirect relationships between entities. GNN is widely used in physics and societal systems, molecular fingerprints, and diseases classification.

4.2 Distributed algorithm running step

Once the CNN module computed the expected best algorithm $a \in A$, the identity of the selected algorithm is broadcasted toward all the robots as well as the positions of the final shape. The two-steps of the CNN based MRSR algorithm are illustrated in figure 28.

4.3 Candidate distributed algorithms

It is obvious that disposing of a pre-processing system, capable of evaluating which algorithm is better based on the initial and the final shape, is an inestimable help. We propose in this chapter a kind of hybrid centralized/distributed method that selects between both algorithms: TBSR and C2SR. The concept is pretty simple and requires two elements: an artificial neural network especially trained to determine the better distributed algorithm according to the self-reconfiguration modular robots global problem: initial shape, final shape, problem size, and so on, this is the centralized part of the contribution. The second element is a bunch of modular robots on which the compared algorithms are preloaded, the decentralized part of the contribution. Where the neural network has evaluated the ideal algorithm, its identifier is diffused to all robots so they can run it.

In this first work, we studied two already published algorithms related to the same modular robots platform 2D-catom [46]. Indeed, even if there is much work on distributed self-reconfiguration algorithms, the multiplicity of modular robots platforms makes difficult to find comparable approaches. For instance, the distributed algorithm in [10] can not be used with 2d-catoms robots, because it assumes that robots evolve over a horizontal surface, while 2d-catoms are stacked on top of each other over a ground support that can not be crossed. In addition, other characteristics make difficult if not impossible the adaptation of a distributed algorithm to other platforms. For example, some works put strains on the accepted initial or the final shapes of the MRSR. Even if we project to adapt some other approaches to 2d-catom robots, we started our study with two published methods: C2SR and TBSR.

Hereafter, a brief description of the two studied algorithms:

- Cylindrical-Catoms Self-Reconfiguration (C2SR): In this distributed algorithm [70], the final shape is constructed, at the right of the initial shape, starting by the ground to the top lines. To do so, the initial shape is emptied starting from the top to the ground (see figure 30.(a)). The rightmost bottom robot of the initial shape does not move and is considered as the leftmost bottom robot in the final shape.

- Translation Based Self Reconfiguration (TBSR): TBSR [13] aims to balance the number of moves made by each robot. To that end, TBSR attempts to keep the left-right positions of the robots within the initial and final shapes. In other words, in TBSR, a robot on the left side (resp. right side) of the initial shape has a high probability to be at the left (resp. right) side of the final shape. The bottom line is constructed first, then TBSR constructs final shape diagonals starting from the rightmost to the leftmost position. On the other side, the initial shape is emptied starting from rightmost to the most left diagonal with keeping the ground line to the end (see figure 30.(b)).

Despite the fact that TBSR exploits the left-right disposition of the robots in the initial and final shapes, a comparative study of both approaches shows that C2SR outperforms TBSR in some cases. Indeed, the mass unbalance between the left and right sides of both initial and final shapes makes it not always possible to conserve the left-right disposition of the robots. Analytical characterization of these cases is a complex task, hence the use of the ANN approach.

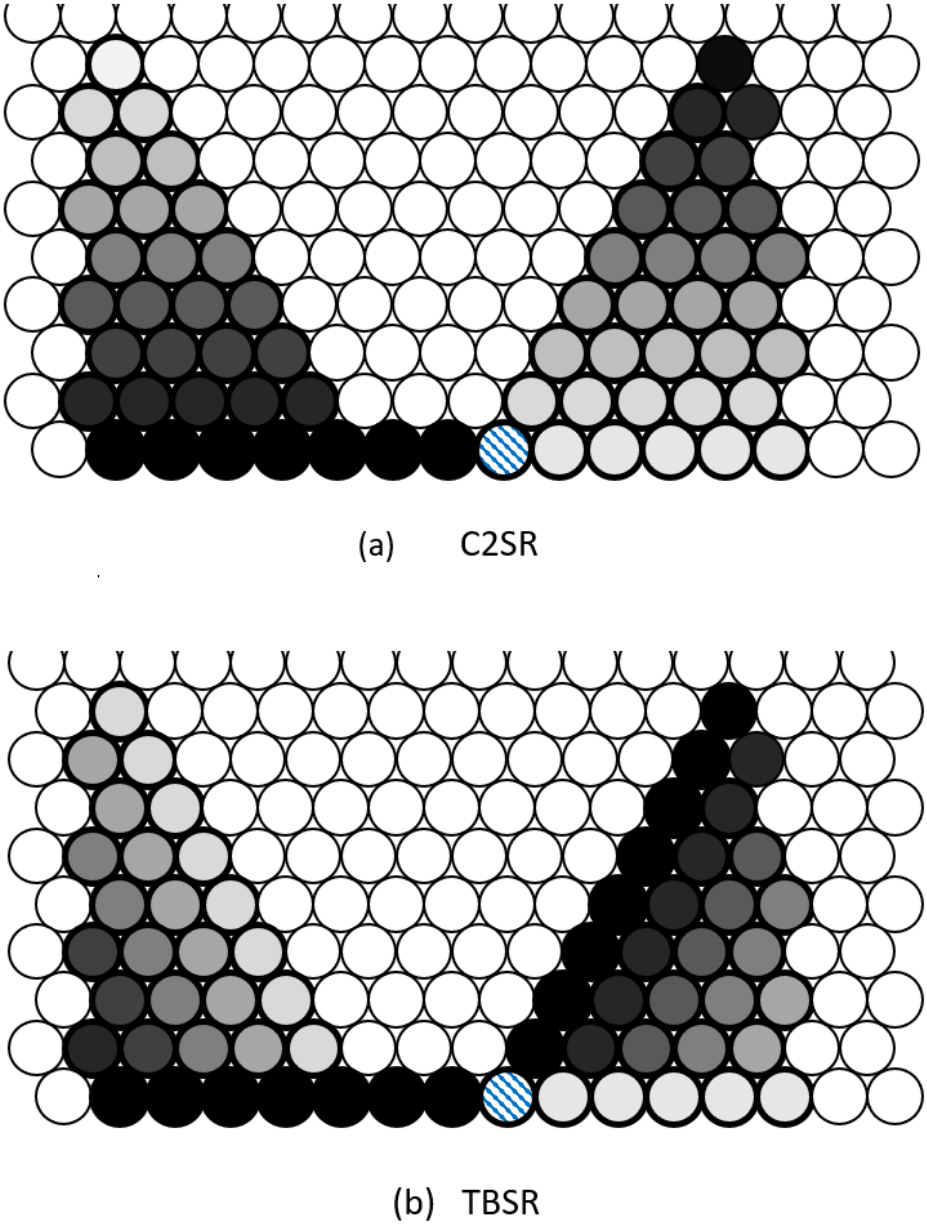


Figure 30: Comparison of C2SR and TBSR working schemes. The hashed position corresponds to the common position between the initial and final shape and represents the separation between the initial shape (at the left of the common position) and the final shape (at the right). The gray level of a robot's position, from the lightest to the darkest, designates the order of deconstruction (respectively construction) of the initial shape (resp. final shape).

The objective of the Neural Network system is to identify the most suitable reconfiguration algorithm to use based on the description of the initial and final shape. In the next section, we discuss the different choices made in the selection and configuration of the ANN.

5 Convolutional Neural Network Modeling

In this section, we detail our Neural Network based modular robot Self-Reconfiguration called CNNSR. Due to the nature of the modular robot self-reconfiguration problem, it is obvious that geometrical relations between the occupied positions in the initial and the final shapes play a key role in the problem characterization. It is therefore legitimate to consider a class of ANN adapted to this kind of problem. As discussed in §4.1, Convolutional Neural Networks (CNN) fit well for spatial relations management. Therefore, we selected CNN for the identification of the best MRSR algorithm.

5.1 NN inputs

The starting point of the description of a reconfiguration problem is the specification of the initial and the final shapes. We opted for merging the initial and final shapes within the same structure that indicates the positions of the modular robots at the beginning and their target positions in the final shapes (see figure 29).

Figure 31 shows an example of the input matrix corresponding to the problem of re-configuring 5 robots initially representing a horizontal line into a diagonal line. The most left position of the robots on the floor in the final shape is considered equal to the rightmost position of the robots on the floor in the initial shape.

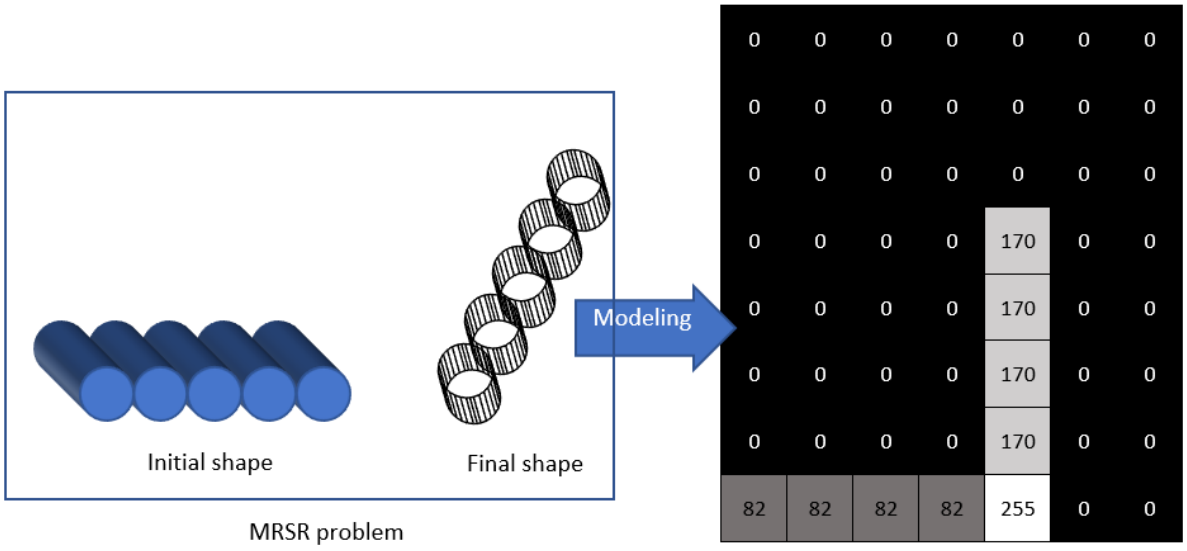


Figure 31: MRSR Problem conversion into an input grayscale image. The space positions are coded by pixels. Pixels in black are the positions out of the initial and the final shapes. Dark gray pixels are the positions occupied in the initial shape but empty in the final shape. The light gray pixels represent the occupied positions in the final shape that are empty in the initial shape. The white pixels determine the positions occupied in both initial and final shapes.

5.2 Convolution matrix

The convolution layer is the main construction feature of a CNN. It consists to apply a sliding filter on the blocks of the layer inputs. The convolution layer inputs, M_i, are organized into 2D matrix $N_i \times N_i$ that correspond in the case of the first convolution layer to the input matrix

described in the paragraph 5.1. The working scheme of the convolution procedure is defined by three parameters: the depth, the step and the padding. The depth designates the size of the filter matrix and therefore the size of convolution blocks also called kernels. The kernel is a sliding square contiguous $K \times K$ sub-matrix of the input matrix M_i. The step S defines the overlapping degree between kernels. A low value of S leads to a high overlapping level between the kernels. Finally, the padding parameter, P, represents the number of zero-lines and columns added to the input matrix M_i borders. We notice M_i^* the input matrix after the padding insertion. Zero padding allows controlling the size of the output matrix M_o of the convolution layer. The working scheme of the convolution layer is described in Figure 34. Each node $M_o(i,j)$ in the output matrix is computed as follow:

$$M_o(i,j) = \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} M_i^*(k + i.S, l + j.S) \times F(k,l) \tag{4}$$

Where F is the convolution matrix also called convolution filter. The output matrix M_o is represented then by a square matrix $N_o \times N_o$ where :

$$N_o = \frac{N_i - K + 2.P}{S} + 1 \tag{5}$$

As we will explain below, we used different kernel sizes, depending on the case, we have $K = 3$ or K = 5.

During the first convolution cycle, the input matrix M_i corresponds to the MRSR problem mapping matrix given in section 5.1. When converted to a matrix, 2 neighboring positions of each pixel won't actually be in contact with it: the bottom-left and the top-right neighbors (pixels 1 and 5 in Figure 32).

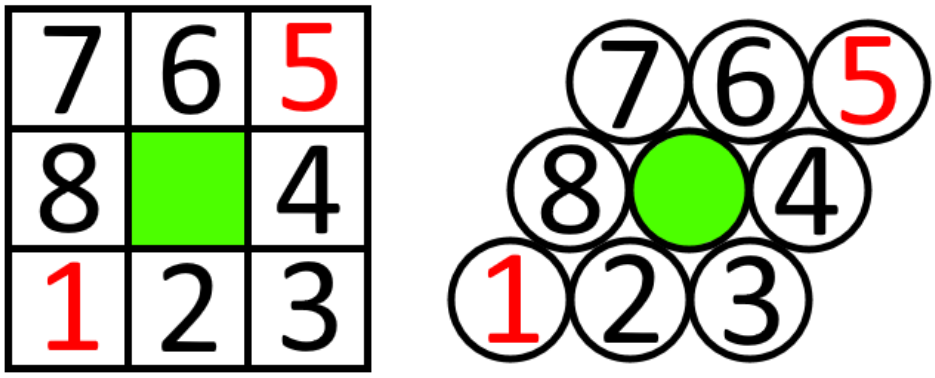


Figure 32: When going from a 6-lattice to a 9-lattice, 2 neighbors are actually connected in reality

Therefore, we studied three filter matrix patterns: F_1, F_2, $F3$ expressed by the following 3×3, 3×3 and 5×5 matrices.

$$F_1 = \begin{pmatrix} x_{0,0} & x_{0,1} & 0 \\ x_{1,0} & x_{1,1} & x_{1,2} \\ 0 & x_{2,1} & x_{2,2} \end{pmatrix} \tag{6}$$

$$F_2 = \begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,0} & x_{2,1} & x_{2,2} \end{pmatrix} \tag{7}$$

$$F_3 = \begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,0} & x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{pmatrix} \tag{8}$$

F_1 and F_2 used a depth parameter $K = 3$. If F_1 filtering pattern is used, the neural network training will not consider the top-right and the bottom-left neighbors during the first convolution cycle.

In F_3 pattern, the convolution process is extended to kernels with a size $K = 5$. This way, the convolution layer obtains a wider vision of the local organization of the robots. Indeed, the move of a modular robot depends on its direct neighbors but also on the second range neighborhood.

The best value of the Filter matrix is deduced from the training stage of the CNN. The step parameter S is fixed to 1, while the padding parameter P is 0. Indeed, since the bottom line is in contact with the flour, adding a zero-pad to the bottom of the image generates confusion with the significance of black pixels, i.e. positions that do not belong to both initial and final shape. Using equation 5, we deduce that the size of the convolution layer output $N_o = N_i - 2$ for filtering pattern F_1 and F_2 and $N_o = N_i - 4$ for the filtering pattern F_3. N_i is the size of the input image.

5.3 NN outputs

The CNN output depicts the expected best-suited reconfiguration algorithm for the given inputs. The output layer of the CNN is made out of two nodes, one for each benchmark algorithm (respectively C2SR and TBSR). The values of those nodes correspond to the suitability that the CNN attributes to each algorithm. 0 means that the approach is bad while 1 value means that it is a good approach. The selected algorithm is determined by the highest value of the two nodes.

6 Experimental results

Artificial Neural Network requires a training phase to learn how to optimize the ability of the neural network to identify the best reconfiguration algorithm according to the input image. The objective of the training and validation datasets is to optimize the convolution filtering matrices and connections' weights of the classification graph.

6.1 Dataset

To do so, we constructed 1208 different MRSR scenarios representing different cases of initial and final shapes (see Figure 33, with varying the number of robots from 11 to 121.

On each problem, we run the C2SR and TBSR algorithms and stored the performances of each algorithm in terms of the number of moves needed to reach the final shape. Then, we classified the different problems into two categories: problems where C2SR performs better, and problems where TBSR performs better. This pre-processing step leads to two sets of 664

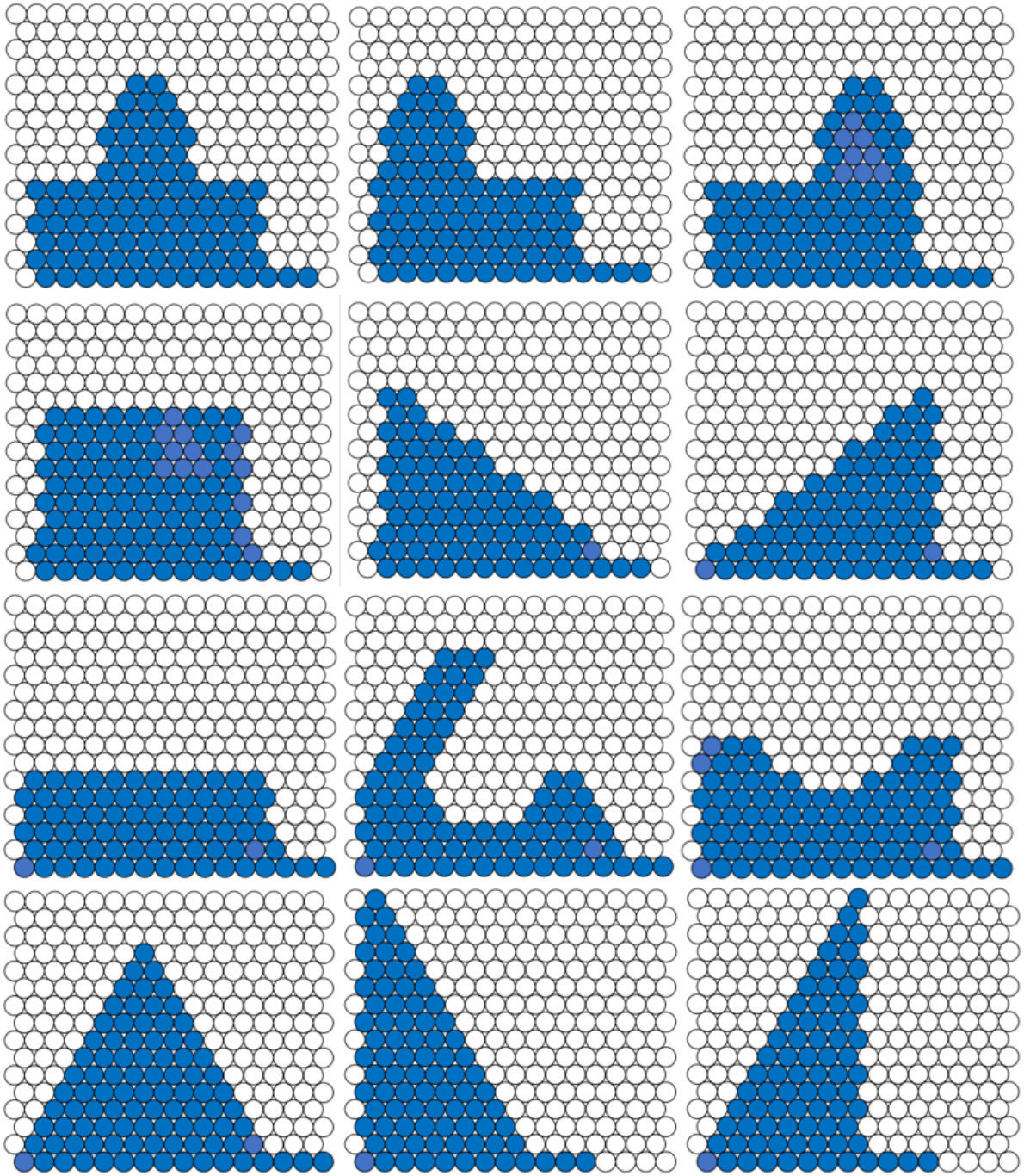


Figure 33: The 12 categories of shapes composing the studied 1208 scenarios. The scenarios are generated using these shapes alternatively as initial or final shape in the MRSR problems.

of scenarios where TBSR is better and 544 where C2SR is better.

80% of the dataset problems were used to train the CNN model thanks to the supervised learning method. 10% of the dataset is used to validate the model. The remaining 10% of the dataset is used for the test and the comparison of the models given hereafter.

The CNN models are implemented using Keras library [18]. Each CNN model is trained during

100 epochs using Adam [49] gradient descent algorithm with a parameter $learning - rate = 10^-3$. A detailed overview of the CNN model is depicted in Figure 34.

In the following paragraphs, we analyzed and compared the different studied CNN models and the impact of the different parameters.

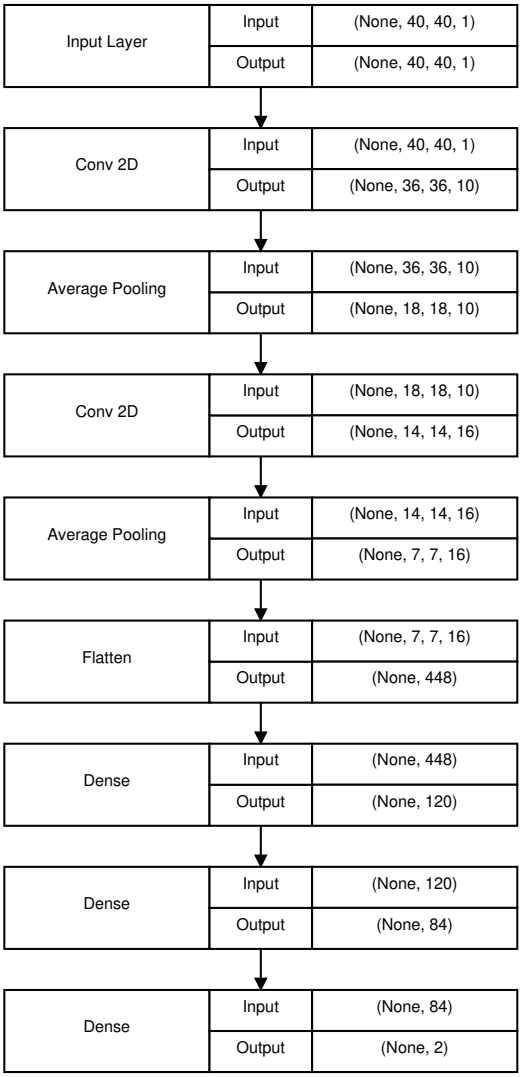


Figure 34: Detailed description of the CNN model

6.2 Impact of the convolution filtering pattern

In section 5.2, we proposed three convolution filtering matrices noticed F_1, F_2 and F_3. The application of the associated CNN model on the 120 self-reconfiguration scenarios composing the test dataset shows that using F_1 filtering matrix leads to an accuracy of 96.67%. This means that the CNN model succeeds in identifying the appropriate distributed algorithm, according to the total number of moves, in 96.67% of the time. The confusion matrix is given in Figure 35.(a). We observe that the neural network presents 3 errors over 52 cases where C2SR is in fact better than TBSR (a precision of 94.23%), while there is only one error over the 68 cases where TBSR is better (98.52%). We deduce that our CNN system is much more trained for

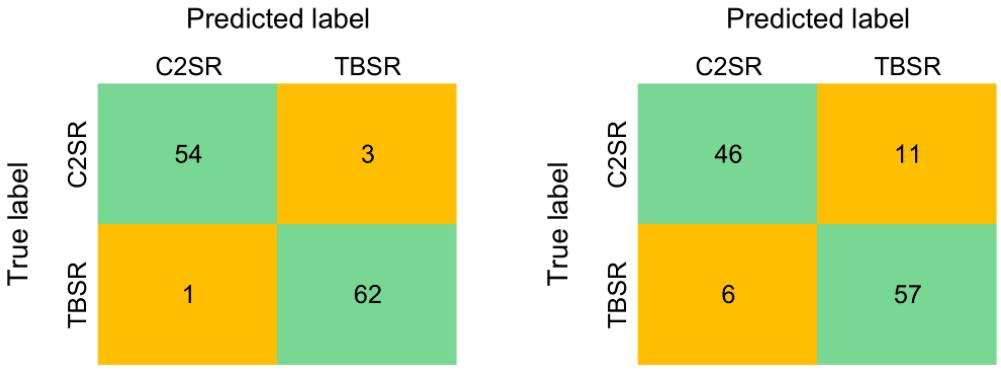


Figure 35: Confusion matrix for 120 test scenarios using filter function (a) F1 (b) F3 and the total number of moves metric

detecting TBSR's adapted scenarios than C2SR's adapted scenarios. This can be explained by the preponderance of TBSR's adapted cases in the dataset. This observation is confirmed with the model using the filtering matrix F_3 (see the confusion matrix of Figure 35.(b)).

The success rate dropped down to 60% when the filtering matrix F_2 is used. We deduce that the information presented by the diagonal pixels are indispensable for the description of a robot's position state. In the other side, the extension of the filtering matrix to the second range neighbor using the filtering matrix F_3 presents a success rate of 86.67% (see Figure 35.(b)). We conclude that the use of the filtering matrix F_1 outperforms the two other models.

6.3 Impact of the used optimality metric

The evaluation of the suitability of a given distributed algorithm in comparison to another one is a difficult task. Indeed, using the total number of move, SUM, as a metric to identify the best distributed algorithm is debatable. Indeed, requiring unbalanced efforts among robots may lead to adverse outcomes, such as a long reconfiguration time, or a global freeze of the execution due to a lack of energy for a subset of robots. Therefore using the maximum number of moves made by a robot, MAX, as a metric for evaluating algorithm suitability may be considered. Standard deviation of the number of moves made by the robots, DEV, provides a good way to estimate the effort equity between robots. However, guaranteeing the effort equity does not ensure the convergence efficiency of the self-reconfiguration process.

In Fig. 36.(a) and 36.(b), we give the confusion matrices of the CNN using the metrics MAX and DEV respectively, since we decided to put SUM aside. We observe that using MAX or DEV metrics instead of the SUM metric leads to more confusion. The success rate decreases from 96.67% with SUM metric to 80% with MAX metric and 89% with DEV metric. The CNN model faces more difficulty to extract the MRSR problem characteristics that make a given algorithm more suitable for reducing the MAX metric. Indeed, the MAX value is unstable value determined by only one robot while SUM and DEV metrics are aggregations of all robots' pattern.

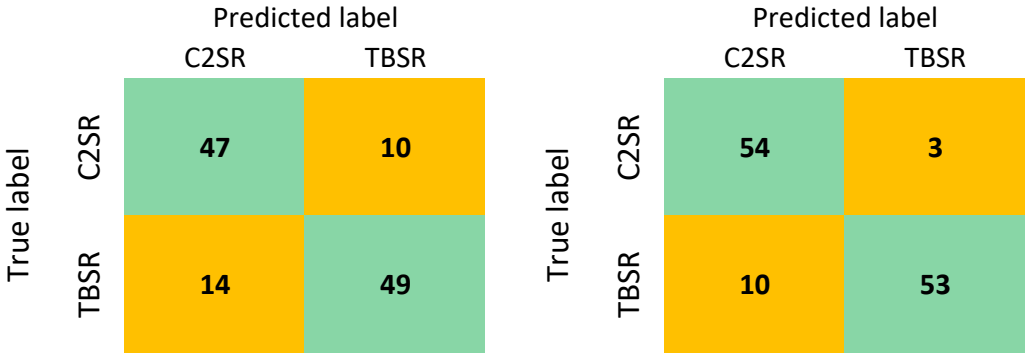


Figure 36: Confusion matrix with 120 test scenarios using the function F1 and (a) the maximum number of moves metric (b) the deviation of the number of moves per robot

6.4 Impact of the CNN outputs pattern

The result of the CNN is a decision corresponding to which method is better: C2SR or TBSR. The coding of this decision can be done in two ways. In the first mode, the output corresponds to two binary nodes. A couple of values (1,0) corresponds to one method, while (0,1) corresponds to the other method. In the second mode, the output is coded by a single binary node that returns 0 when C2SR is expected to be better and 1 if it is TBSR.

The comparison results show that the two models are equivalent in terms of the success rate 96.67% using F_1 matrix).

6.5 CNN overall performances

The results of the final CNN system over 120 MRSR scenarios show a low error risk (3.3%) in the identification of the most suited self-reconfiguration algorithm. The analysis of the 4 test scenarios leading to the CNN error shows that these scenarios are all characterized by highly similar results of C2SR and TBSR algorithms. Therefore, we conclude that the risk of significant efficiency lost due to a bad selection of the distributed algorithm is very low. In figure 37, we show an example of MRSR problem for which the CNN model returns TBSR algorithm whatever is the used metric: SUM, MAX and DEV. The execution of the TBSR and C2SR on the problem shows that this choice is right in the three cases.
More generally, the obtained results prove that the Artificial Neural Network is an efficient approach for MRSR problem classification. One issue is thus to predict our CNN behavior when more distributed algorithms are pre-loaded on the robots. It is expected that when the number of distributed algorithms increases, the ratio of confusion cases increases too. However it is obvious that the seriousness of the mis-classification is lower. Indeed, as the number of algorithms increases, similarities in terms of algorithms' strategies and performances will appear. This makes more difficult to the CNN to determine the best algorithm, but it remains strongly expected that the returned algorithm be a good one. In this work, the CNN output is a binary couple (x_1, x_2) that determine either the first $((x_1, x_2) = (1, 0))$ or the second algorithm $((x_1, x_2) = (0, 1))$ is selected. The CNN model could be easily extended to n algorithms by coding the output as a vector $(x_1, .., x_n)$ where $x_i = 1$ if the i^{th} algorithm is selected and $x_i = 0$ otherwise.

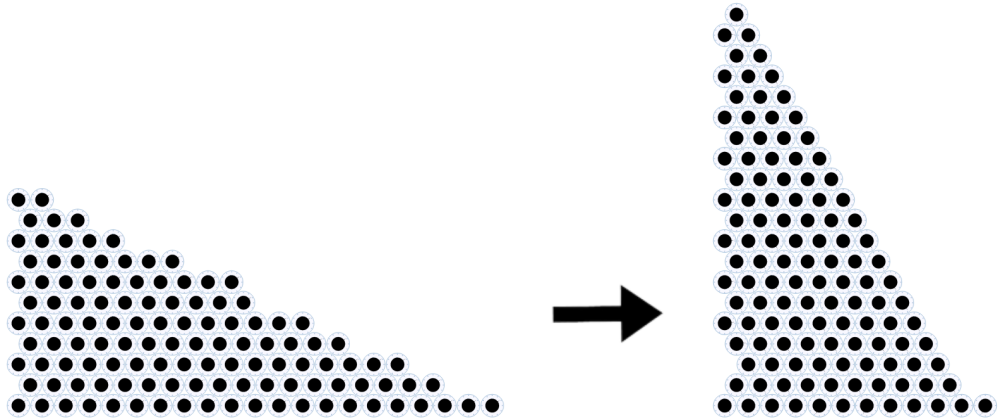


Figure 37: Example of a MRSR problem. Returned algorithm by CNN is TBSR. The SUM, MAX and DEV performances of C2SR are 2854, 41, 42.25 while the performances of TBSR are 2714, 33, 41.84.

7 Analysis of the CNNSR approach

Self reconfiguration of modular robots is a very challenging problem. Despite ample literature on the self-organization algorithms, a clear comprehension of the efficiency conditions of each algorithm remains lacking, preventing the reuse and the combination of these methods.

The objectives of the CNNSR approach are twofold. First, an original hybrid approach combining the centralized and distributed reconfiguration methods is proposed. Secondly, the approach provides an ideal framework to exploit any distributed self-reconfiguration algorithm. The hybrid approach starts with a centralized phase consisting of a CNN system that analyses the MRSR problem in order to identify the most appropriate distributed self-reconfiguration algorithm. Then the robots are notified by the desired final shape and the recommended self-reconfiguration algorithm to use. In the second phase, the selected distributed algorithm is executed over all the modular robots to build the target shape.

This first work discusses many variants of the CNN model and compares their results using a dataset of 1204 scenarios. Tests show that the CNN system is able to identify the appropriate distributed algorithm with a very high precision (96.67%).

The presented work is a first, but important step in the path towards intelligent modular robot systems. Other improvements of the work are necessary. The quality of a given distributed algorithm is measured according to the number of required moves to reach the final shape. However, the nature of the move (horizontal, up, down) is not considered while the energy consumption depends on the move nature. In addition, it is important to consider the multi-criteria nature of the MRRSR problem by aggregating different metrics such as the total number of moves, the maximum number of moves per node, the maximum amount of energy consumed per node, and the deviation of the effort over the nodes.

In terms of neural network modeling, CNNSR approach does not distinguish between the cases where one of the distributed algorithms is significantly better than the other and when both algorithms are quite similar. In this way, the CNN will penalize the misclassification of TBSR-

adapted problem into C2SR and vice versa. The next section is dedicated to the CNN2SR version of our CNN-based self-reconfiguration approach that classifies the problems into TBSR-adapted, C2SR-adapted and Neutral categories.

8 Fine-grained artificial intelligence approach - CNN2SR

CNN training phase tries to optimize the Neural Network in order to reduce the misclassification errors. CNNSR proceeds by a strict classification of the problem into either TBSR or C2SR adapted problem. The evaluation of the CNN model during the training phase is done according to the error rate. This error rate is computed as the number of cases where a TBSR-adapted instance is considered a C2SR-adapted instance and vice-versa. However, this evaluation does not take into account the level of severity of the error. In other words, is the instance significantly more easily solved by either method or do both provide similar results? In this second variant of the CNN-based self-reconfiguration method, called CNN2SR, the classification of the instances is made according to three categories: TBSR-adapted problem, C2SR-adapted problem, and Neutral.

This section is organized as follows: subsection II gives an overview of the MRSR methods proposed in the literature and then explains the hybrid method, especially its Artificial Neural Network. In subsection III, we present the different components of the implemented Convolutional Neural Network (CNN). subsection IV gives details of the CNN training process, the used dataset, and the obtained results. Finally, in subsection VI, a conclusion of this work is given.

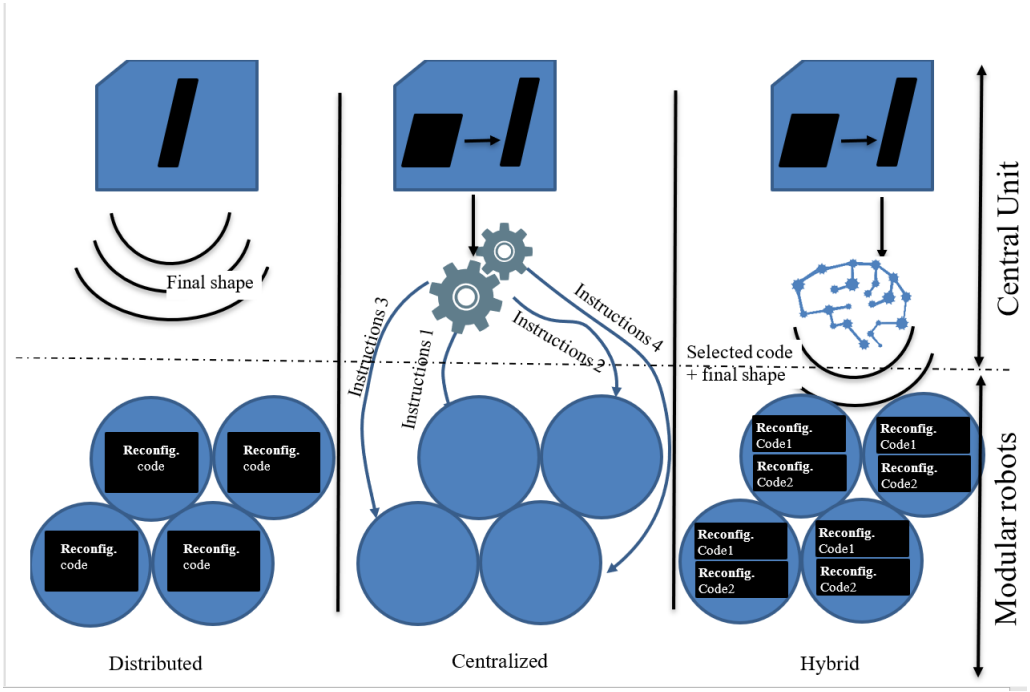


Figure 38: Distributed, centralized and hybrid approaches working schemes. Disks represent the modular robots. Communication between the central unit and the modular robots is either unicast (centralized approach) or multicast.

8.1 Neural Network-based Modular Robots Self Reconfiguration

CNN2SR represents an improvement of the previous CNN model for the modular reconfigurable robots self-reconfiguration problem, CNNSR. Some characteristics of the CNNSR are then kept unchanged and others are modified to introduce the neutral category of classification.

8.1.1 CNN inputs

We adopted a CNN model and assimilated the MRSR initial and final shapes to an image as it is shown in Figure 39. Each potential position is represented by a pixel (a cell) in the image. Dark pixels correspond to the positions occupied initially by the modular robots and should remain occupied in the final shape. Light gray pixels indicate the positions occupied initially but do not belong to the final shape. Dark gray pixels represent the positions that should be occupied in the final shape but not initially occupied.

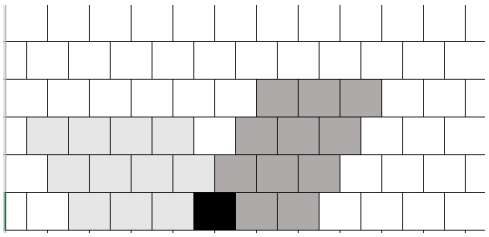


Figure 39: Representation of the initial and the final shapes.

8.1.2 CNN output

The CNN output, o, corresponds to a triplet $(p_{C2SR}, p_{Neutral}, p_{TBSR})$. The value p_x designates the estimated probability that the problem belongs to the class: C2SR, Neutral, and TBSR. The class C2SR (resp. TBSR) means that C2SR algorithm (resp. TBSR) is well suited for the problem. The neutral class means that the two methods are slightly equivalent. C2SR and TBSR are considered equivalent if the expected performances of C2SR, $Cost_{C2SR}$ and TBSR, $Cost_{TBSR}$ respect the following condition:

$$\frac{|Cost_{C2SR} - Cost_{TBSR}|}{\max(Cost_{C2SR}, Cost_{TBSR})} \leq 0,01 \tag{9}$$

The cost of a given method represents the expected number of moves required to reach the final shape using the corresponding method.

8.1.3 CNN implementation

To implement our CNN model We used Keras library [18, 92]. The CNN model is composed of 10 layers as shown in Figure 40.

The MRSR inputs are coded by 48×48 gray-scale images (see Figure 39. The CNN model includes 3 cycles of convolutional layer, $Conv2D$, and pooling layer $MaxPooling2D$. The three convolutional layers correspond to respectively 8, 16, and 32 filters and use the activation method $ReLu$. The convolution matrix corresponds to matrices of 3×3.

The resulting matrices are passed as inputs to a Flatten layer that transforms the multidimensional array $4 \times 4 \times 32$ in a unidimensional array with 512 values.

This last block reduces the dimension of the resulting array using three fully connected Dense layers composed of, respectively, 128, 32, and 3 neurons. The last layer output is composed of three neurons that represent the problem classification probabilities: p_C2SR, $p_{Neutral}$, p_{TBSR}. $Softmax$ function is used as the activation function for the last Dense layer.

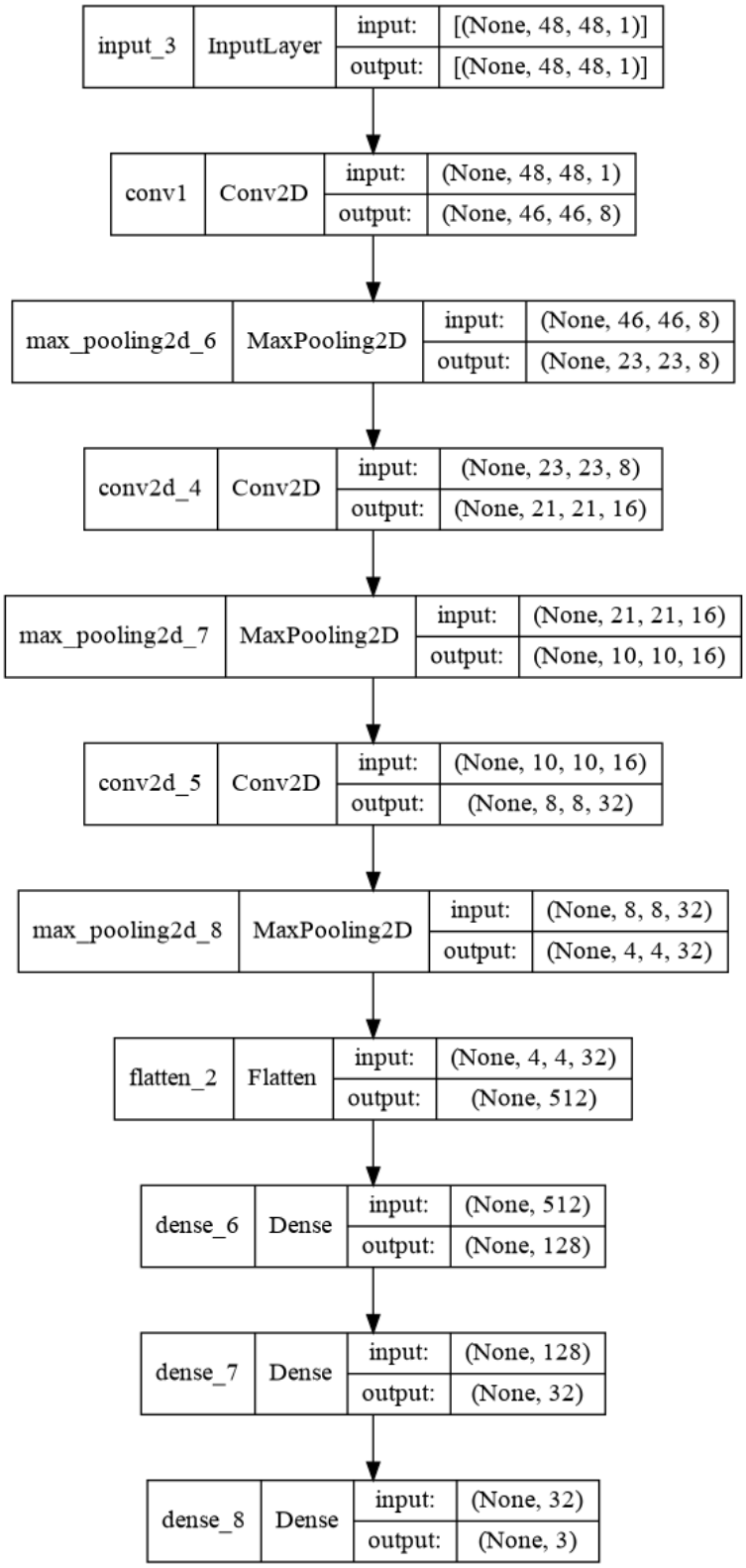


Figure 40: Multiclass classification neural network architecture details.

To determine the class of the input problem, the highest value between p_C2SR, $p_{Neutral}$ and p_{TBSR} is considered.

8.2 Experimental results

8.2.1 Dataset

The input data of the neural network corresponding to the initial and final shape of the MRSR problem are represented by an image (see subsection 8.1.1). For the needs of the CNN training, validation, and test, we choose 12 basic shapes depicted in Figure 33. Each basic shape is available in three different sizes: small with 11 robots, medium with 51 robots, and large with 121 robots. In addition, each basic shape is translated using different geometrical transformations (90°, 180° and 270° rotation). All MRSR images are stored in gray-scale Bitmap 48×48 format.

3645 MRSR problems are generated using the 12 different basic shapes sometimes playing the role of the initial shapes, and sometimes as final shapes. 80% of the dataset's problems are used for the CNN training while 10% are considered for the validation task (model improvement). A sample of 10% (364 scenarios) of the dataset is held back from the training and validation phases and is dedicated to the final performance analysis. The following results of the CNN model are obtained using this test sample.

For the purposes of CNN training, C2SR and TBSR have been run on all the dataset scenarios. The dataset is then classified into three categories (sub-sets) according to the obtained results: C2SR scenarios, Neutral scenarios, and TBSR scenarios.

To improve predictions as well as the classification accuracy of the model, "one hot encoding" is performed. To each scenario, x, we have associated a label vector, $L(x)$, of 3 values. When C2SR algorithm is more suited to the scenario x, $L(x) = [1, 0, 0]$. Neutral scenarios are associated to a vector $L(x) = [0, 1, 0]$ and TBSR scenarios to a vector $L(x) = [0, 0, 1]$. The objective of neural network training is to maximize the matching between the CNN output $[p_{C2SR}, p_{Neutral}, p_{TBSR}]$ with the associated labels of the scenarios.

8.2.2 Neural Network performances

Neural Network model optimization follows an iterative cycle called epochs. During each epoch, the parameters of the CNN model are first optimized to match the training scenarios' labels, $L(x)$, with the CNN outputs $[p^x_{C2SR}, p^x_{Neutral}, p^x_{TBSR}]$. Then the obtained CNN model during the current epoch is evaluated based on the validation dataset. The performances of the Neural Network are evaluated according to two metrics: loss and accuracy. The accuracy measures the ratio between the number of correct predictions of the scenario's class and the number of scenarios.

$$accuracy = \frac{\sum_{x \in X} \left(L(x) \stackrel{?}{=} [p^x_{C2SR} \stackrel{?}{=} M_x, p^x_{Neutral} \stackrel{?}{=} M_x, p^x_{TBSR} \stackrel{?}{=} M_x] \right)}{|X|} \tag{10}$$

M_x represents the highest value among p^x_{C2SR}, $p^x_{Neutral}$, and p^x_{TBSR}. The expression $a \stackrel{?}{=} b$ returns 1 if the equality is true and 0 otherwise. Training accuracy refers to the accuracy rate when X is the training dataset. Validation accuracy refers to the accuracy rate when X is the validation data set.

Furthermore, the loss function measures the error degree between the predicted output of the CNN model and the precomputed labels of the scenarios. We used the cross-entropy method discussed in [69].

Figure 41 shows the evolution curves of the accuracy and loss over the epochs on both training

and validation data sets. As expected, the performances of the CNN model are slightly better on the training dataset than on the validation data set. Indeed, the training data set are used to optimize the performances of the CNN, while the validation data set is only used for evaluation purposes. We also observe a stagnation of the CNN improvement at the end of the training/validation process (after 30 epochs).

A naive analysis of the network's performances shows a respectable accuracy of 90.66%. The

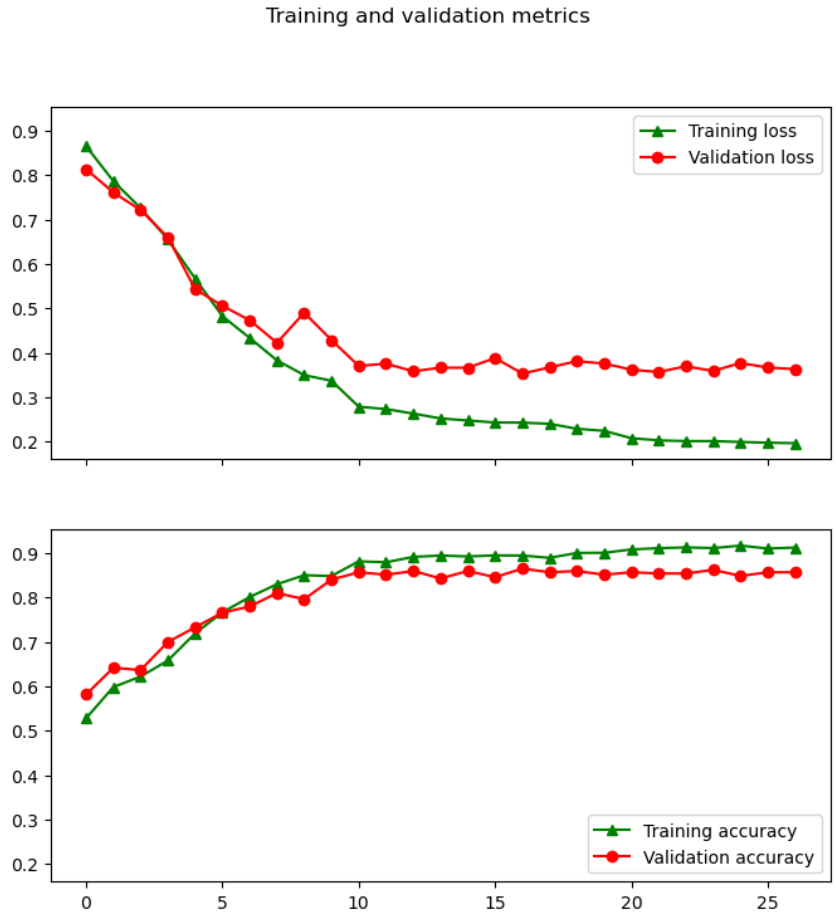


Figure 41: Evolution over epochs of both training and validation quality metrics. In the top, the loss evolution and in the bottom, the accuracy evolution.

,

computed confusion matrix (Figure 42) confirms this accuracy score and details the prediction results over the test data set. Rows correspond to the true labels of the test scenarios, while the columns refer to the predicted classes (0=C2SR; 1=Neutral, and 2=TBSR). The cells' values, in the confusion matrix, indicate the number of test scenarios of a given class (the row index) classified by the CNN as belonging to a given class (column index). The accuracy score corresponds to the ratio between the sum of diagonal cells and the size of the test dataset.
However, a neutral scenario is a case for which C2SR or TBSR performances are equivalent. Therefore classifying a neutral scenario as TBSR or C2SR scenario is not a big issue. In light of this, we have modified the confusion matrix (Figure 43) and have moved the Neutral to TBSR conversion and Neutral to C2SR conversion into Neutral to Neutral cases. With this consideration, the CNN accuracy reaches 97.25%.

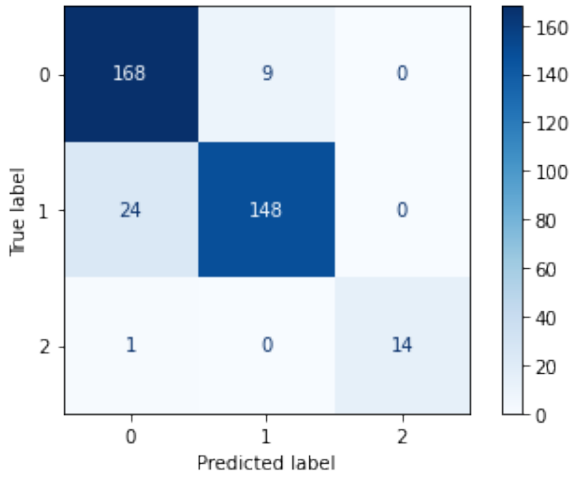


Figure 42: Training and validation metrics curves giving an overview of the training step of our model.

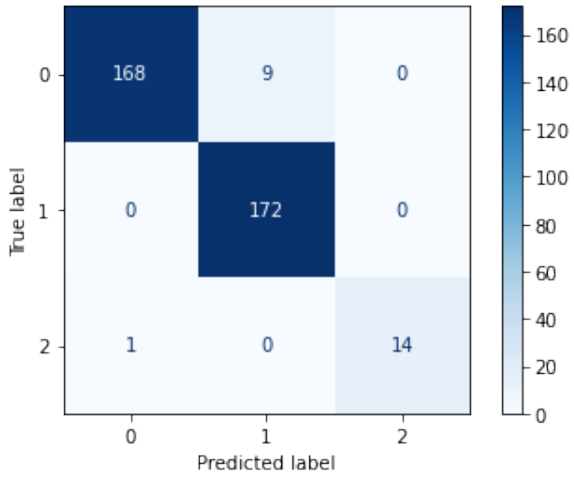


Figure 43: Training and validation metrics curves giving an overview of the training step of our model.

There are nonetheless a few errors left as we can see on the confusion matrix (10 over 360 test scenarios). There are two types of error in our context. The most serious errors, called inversion error, concern the wrong classification of TBSR scenario as C2SR, or vise-versa. Wrong classification of TBSR or C2SR scenario as neutral is less serious. For our test subset, the class inversion error happened only once, as we can see in the confusion matrix of Figure 43, which corresponds to a probability of 0.27%. The neutralization error happened a little bit more, 9 times actually, and presents a probability of 2.47%.

8.3 Conclusion and perspectives

The use of a Neural Network system to select the best modular robot self-reconfiguration algorithm is studied. The idea is to determine based on the initial and the final shapes, the most suitable distributed algorithm to use. This approach provides a way to combine, at a lower cost, different distributed algorithms from the literature.

The CNN model is adopted due to the relevance of such a neural network with image processing problems. The obtained results show that the CNN model presents an accuracy of 97.25%.

The presented model could be adapted to any number of different algorithms in the future, and so, become a strong tool for the modular robotics research community. Also, the dataset needs to be updated with other basic shapes to improve the neural network scope.

This tool still can be improved, mainly by creating a custom loss function instead of using the classical categorical cross-entropy. This way, the network will distinguish the different types of errors according to their importance. Indeed, the class inversion error is more critical than the neutralization error. Therefore, the correction of the results becomes unnecessary and the CNN is better optimized.

Chapter IV
Energy-aware approaches

Despite the growing popularity of Modular Robot Systems, the challenge of finding an efficient solution remains. In particular, the optimization of the energy consumption in light of the battery's state of charge. This work represents the first attempt to take into account the residual energy of modules and their harvesting capabilities during the reconfiguration process. We introduce a new environment to evaluate energy consumption according to the nature of the move and the exchanged messages. Moreover, we also introduce in this chapter a new variation, called CNN3SR, of our Artificial Neural Network based on this energy consumption environment to select the most efficient solution for each self-reconfiguration scenario.

1 Introduction

When the efficiency of self-reconfiguration approaches are compared, the energy consumption tends to be neglected but in real-life applications, the impact of the residual energy over the modular robots outweighs all the other criteria. It is essential to recognize that the convergence of the robots to the final shape depends on the state of charge of their respective batteries. Once a battery becomes depleted, the robot becomes immobile causing a breakdown in the reconfiguration algorithm which may not converge anymore.

To the best of our knowledge, the approach proposed here is the first attempt to study the impact of the energy state of charge on the feasibility and efficiency of shape-shifting algorithms with the use of artificial intelligence. The proposed method lays on the same hybrid approach, proposed in the previous chapters, based on an artificial neural network. The ANN selects the adapted distributed algorithm regarding the adequacy of its reconfiguration strategy and the initial state of charge of robots' batteries.

The contributions of considering the initial state of charge of the batteries, in the selection of the distributed reconfiguration algorithm, are dual:

- The ANN detects if the considered reconfiguration instance is feasible regarding the considered set of reconfiguration algorithms. Therefore, the algorithms that may lead to blocking situations are discarded. If no algorithm is adequate, the reconfiguration is not triggered preserving the system from unnecessary wastage of energy.
- If several algorithms are applicable, the ANN system may select the best algorithm according to both efficiency and energy-based criteria.

In the remainder of the chapter, we start with an overview of energy awareness in the Self-Reconfiguration Modular Robots literature. Then in section 3, we describe the energy model of 2D-Catom robots according to the nature of the motion and the number of produced connections/disconnections. In section 4, we detail the structure of the used Artificial Neural Network, in particular the inputs and outputs formats. Section 5 describes the experiments and the analysis of the results. Conclusion and future works perspectives are given in Section 6.

2 Related Works

Over the past decade, energy awareness has emerged as one of the major concerns in the Information and Communication Technology (ICT) fields. The issue of energy consumption has

garnered a substantial amount of attention, with a notable emphasis on distributed systems as a whole and more notably, telecommunication networks. This now heightened interest can be seen with the tremendous amount of research works [20, 44, 66] dedicated to addressing those specific energy-related challenges.

The energy issue has been from a very early stage identified as one of the major metrics in the evaluation of modular robots reconfiguration efficiency [76]. However, earlier works on self-reconfiguration of modular robot systems ignore the energy issue [68, 93]. Later, many recent works deal with the optimization of the energy consumption of the robots [25, 32, 53, 64]. Major works in this field focus on the reduction of the total amount of energy needed to achieve the system's reconfiguration [53], which corresponds to the total number of moves made by all robots. Few works tried to consider more relevant metrics for evaluating energy consumption. In [13], authors proposed a self-reconfiguration algorithm that tries to spread the reconfiguration effort over the robots to reduce the deviation of the number of moves made by each one.

Whatever the modular robot hardware platform, the self-reconfiguration algorithms in the literature are all characterized by the following three drawbacks:

- The energy consumption modeling of the robot's elementary motion is basic and considers that all motions spend the same amount of energy [95]. However, the energy cost of an elementary motion depends on the number of involved robots, i.e. the number of disconnections and connections to achieve, and the nature of the motion (going up, going down, rolling over, sliding under). These parameters are depicted in figure 44, which shows 6-lattice modular robots called 2d-Catoms. For example, the red robot rolls over three robots and then disconnects from robots C and D and reconnects with D (using another connector) and E. Figure 45 gives an exhaustive illustration of the different types of motion.

- Self-reconfiguration algorithm assumes that modular robots are, initially, equally charged. Therefore the self-reconfiguration ignores the disparity between different robots' motion capacities. The battery expiration is assumed as a part of the fault tolerance process and should be managed separately by curative mechanisms.

- The battery depletion is not envisaged during the self-reconfiguration procedure. Indeed the self-reconfiguration algorithm assumes that robots are continuously powered by any external continuous current such as wire [74], harvesting system [22], or power transfer between robots [50]. Otherwise, we assume that the reconfiguration process is interrupted until the modular robots are recharged.

This chapter presents, to the best of our knowledge, the first attempt to take into account the state of charge of the modular robots in the self-reconfiguration procedure. The objective is to provide a centralized pre-computing procedure that determines the most suitable self-reconfiguration algorithm to use according to the initial and final shapes and the initial residual energy of each robot.

3 Energy cost model for 2D-Catom modular robots

2D-Catom robots are still in the prototype and demonstration stage. Therefore, a machine learning process based on real-world experiments on self-reconfiguration scenarios requires a high number of modular robots, excessive time duration, and a high risk of error. A first simulation-based phase is more efficient to train an artificial intelligence in order to determine

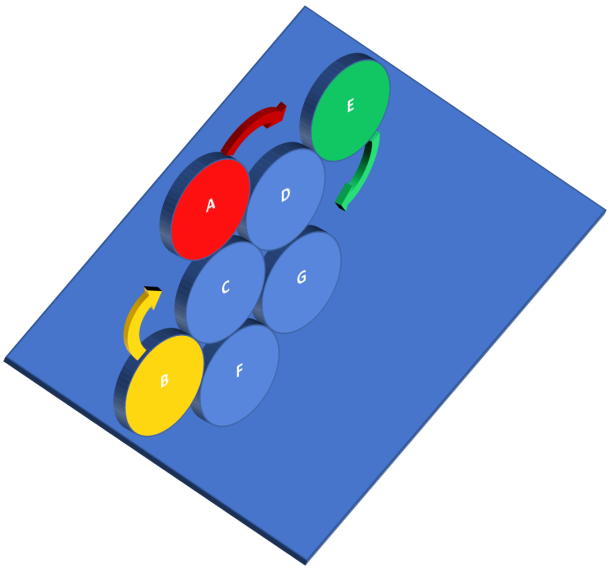


Figure 44: Birds-eye view on 6-Lattice modular robots (cylinders) deployed over a horizontal plane. The motions of three robots (red, green, and orange) are given by three arrows of a corresponding color. The energy cost of each robot is measured according to the number of disconnections and connections to achieve, and the nature of the motion.

the best suitable self-reconfiguration algorithm according to the initial and final shapes.

As the production of 2D-Catoms stills in its first steps, the obtained results couldn't be subjected to comparison with real experiments to experiment and verify the algorithm's effectiveness. VisibleSim [79] is chosen due to its realism, and the simulator is updated frequently by the same teams behind the 2D-Catoms [2]. However, for the needs of our study, we need to involve an energy cost model that simulates the energy consumption and the variation of the battery level of the modular robots.

In this section, we describe the energy model we used to take into account energy constraints in modular robots' self-reconfiguration process. We, first, present the data that model the energy constraints then we expose the energy patterns used in our simulations.

3.1 Energy model

The energy consumption of modular robots depends on the nature of the achieved motion and the number of disconnections and connections done with the neighboring robots. Therefore, a robot may consume energy even when it is still immobile. This is the case when the robot serves as a pivot for the motion of another robot or when the robot disconnects or connects to the moved robot. **We assume that before moving, a robot checks at the same time if its residual energy and the residual energy of involved robots are sufficient.**

To implement that new constraint, we associate the following values to each robot:

- e_i represents the actual battery's residual energy of the robot i.
- e^{max} represents the maximum energy capacity of robots' batteries. This value is assumed equal for all the robots (robots are homogeneous).

- c^{mr} designates the energy cost required for receiving a message from a neighboring robot.
- c^r defines the energy cost required to perform an elementary move/rotation. An elementary move consists in rolling by 60◦ around another robot called "pivot".
- m_b and m_r define two factors applied to c^r to assess the additional energy cost when the elementary move corresponds to climbing (m_r) or moving without a robot below (m_b) (without support, the robot needs more energy to prevent a fall).
- T defines the time, given in seconds, required to achieve an elementary move.
- ci defines the energy spent when a robot stays in the same position (idle mode) during T seconds without exchanging messages and with only turned-off electrodes (no neighbor).
- c_e represents the cost of turning on or off one single electrode

Figure 45 depicts the four kinds of moves that a robot may perform. Robot B performs a simple move that costs c_r. Robot A climbs which costs $c_r * m_r$. Module C moves without support which costs $c_r * m_b$. Module D climbs without support and consumes $c_r * m_r * m_b$. Therefore, the energy cost spent by a robot i to perform a move is computed as follows:

$$c_i^{motion} = \begin{cases} 0 & \text{immobile} \\ c^r & \text{simple move} \\ m_r \times c^r & \text{climb} \\ m_b \times c^r & \text{move without support} \\ m_b \times m_r \times c^r & \text{climb without support} \end{cases} \quad (11)$$

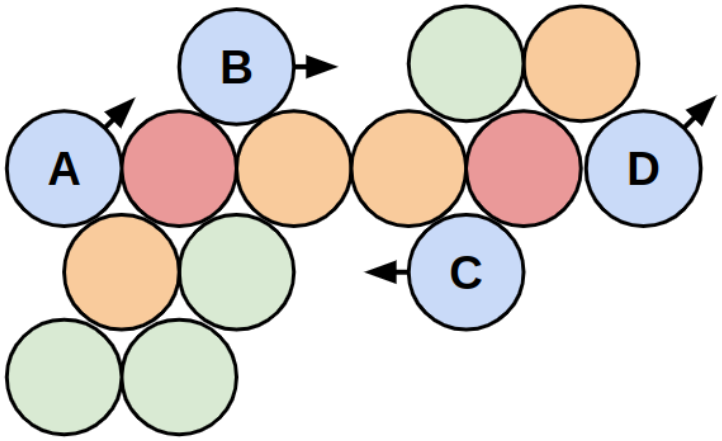


Figure 45: Energy cost of a move depends on the nature of the move: simple move, climbing, move without support, and climbing without support

In the simulations, we have fixed m_r to 1.5, making a climbing move consumes 50% more energy than a simple horizontal move. We have set m_b factor to 2.0, which means that moving without below support costs 2 times more than a simple move.

Equation 12 computes the amount of energy spent by a robot during T seconds, C_i^T, according to the robots' mobility, communications, and connections.

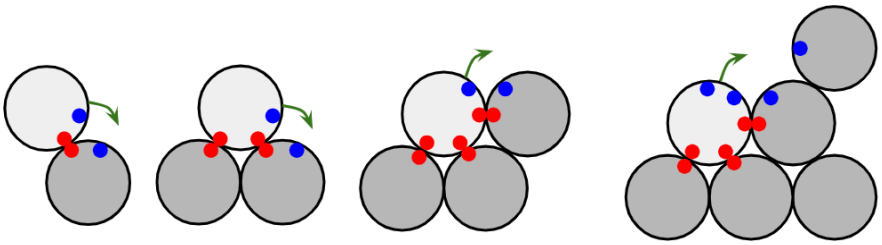


Figure 46: Number of electrodes used in different situation: (a): 4 different electrodes used, 2 to activate and 2 to deactivate (b) 6 electrodes (c) 8 electrodes and (d) 10 electrodes

$$C_i^T = ci + c_i^{motion} + l \times c^e + m \times c^{me} + n \times c^{mr} \tag{12}$$

l, m, and n represent respectively the number of activated electrodes, the number of sent messages, and the number of received messages during the period T. Figure 46 shows how many electrodes are used for different kinds of moves. In a hypothetical case where a robot is isolated, n will equal 0 and C will equal c_i.

We used the data provided in [78] and [45] to set the values of all the variables with realistic values. With this methodology, c_i is worth 286 nanoWatt (nW) and c_e is 533 nW. A robot with a full battery will have 7,4 μWh. The cost for a robot to send a message to a neighbor is negligible when compared to a single move, the orders of magnitude of these two costs are so large (and even larger with the full battery) that we chose to ignore the cost of the messages in the simulation.

3.2 Energy pattern

We defined 15 different energy patterns to use, pictured in Figure 47. Each pattern describes the cartography of the robots' states of charge before the self-reconfiguration. The color of a cell depicts the initial state of charge of the robot at this position (the position may be initially empty). The gray level refers to the residual energy level. Darker colors mean exhausted batteries and light points represent the charged batteries. Those patterns are applied to every self-reconfiguration scenario (initial and final shapes). Therefore, the used energy pattern defines the initial values of e_i for every robot.

In Figure 48, we give an example of MRSR problem represented by an image. Black cells represent the positions that do not belong to the initial shape or the final shape. The other cells are colored in RGB mode. Final positions are colored in red ($RGB =< 255, 0, 0 >$), while initial positions are colored by $RGB =< 0, 255, X >$. The blue level, X, depicts the state of charge of the robot located at the associated position. If the position is both initially and finally occupied, the position will take the color $RGB =< 255, 255, X >$, where X is the initial state of charge of the immobile robot.

4 The artificial neural network for CNN3SR

4.1 Dataset

The scenarios dataset are generated using 12 different classes of shape composed of 120 robots. Figure 33 illustrates these different shape classes. Using these shapes as initial and final shapes,

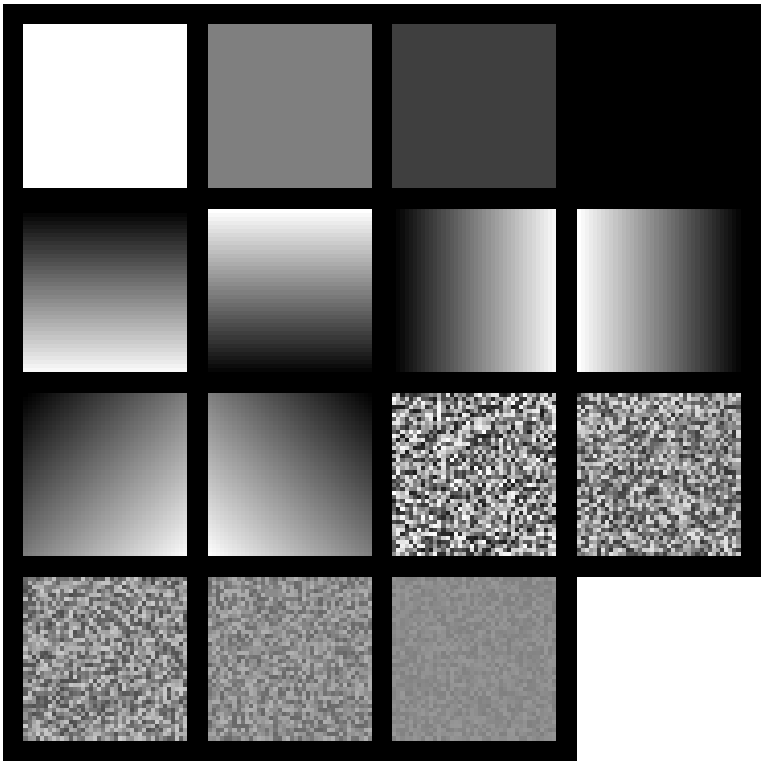


Figure 47: The 15 energy patterns describing the initial residual energy cartography over the initial positions. A white pixel means a fully charged battery (100%) a black pixel means a dead battery. The patterns are: all robots are fully charged, all robots are mid-charged, all robots are charged at 25%, all robots are charged at 0%, horizontal gradient from top to bottom and from bottom to top, vertical gradient from left to right and from right to left, diagonal gradient from top-left to bottom-right and from top-right to bottom-left, and finally, 5 random patterns with residual energy respectively varying in [10%-100%], [20%-90%], [30%-80%], [40%-70%],[50%-60%].

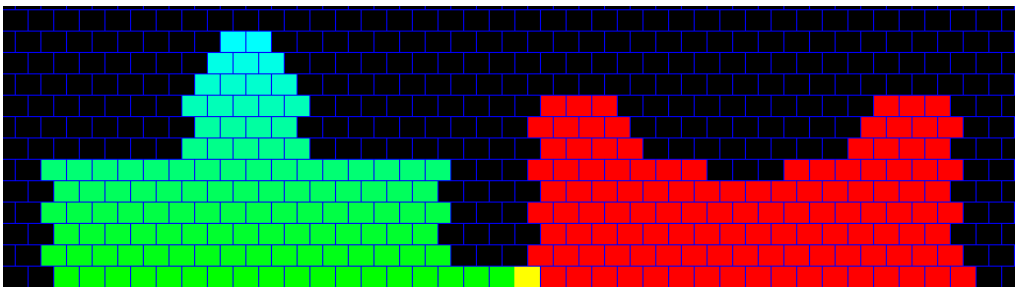


Figure 48: Example of a scenario, here the initial network has the bump1 shape, with the energy pattern no6 (gradient, the top robot has more energy than those on the bottom) into the vase shape.

we obtained 140 different scenarios. For each scenario, we applied the 15 energy patterns described above on the initial shape to obtain 2096 self-reconfiguration scenarios. The objective of the neural network module is to analyze the self-reconfiguration problem in order to predict which distributed algorithm is better for a specific scenario. As explained above, the self-reconfiguration problem is modeled by an RGB image that codes the positions of the initial and the goal shape as well as the initial energy for the battery of each of the robots (see Figure 48). Our dataset scenarios represent 2096 different images of $40px \times 30px$, one image by scenario.

4.1.1 Structure

In this section, we detail our Neural Network-based modular robot Self-Reconfiguration called CNN3SR. Due to the nature of the modular robot self-reconfiguration problem, it is obvious that geometrical relations between the occupied positions in the initial and the final shapes play a key role in the problem characterization. It is therefore legitimate to consider a class of ANN adapted to this kind of problem. Convolutional Neural Networks (CNN) are widely used for image and video processing due to their ability to identify the spatial relation between pixels [26, 37]. Therefore, we use a CNN to identify the best MRSR algorithm by analyzing the image that represents to the self-reconfiguration scenario as illustrated in Figure 48.

Figure 49 gives the details of the adopted CNN architecture. Mainly, CNN is based on four cycles of convolution and pooling then 3 layers of classification. The last layer represents the output layer composed of two nodes respectively associated to the TBSR and C2SR scores.

4.1.2 Outputs

Energy optimization raises a major question about how to evaluate the energy efficiency of a MRSR algorithm. In this study, we identified 3 criteria to judge the energy efficiency of a self-reconfiguration algorithm. This study represents the first time advanced energy efficiency criteria are used for MRSR problem including an energy model for the different kinds of moves.

1. Sum of the energy used by all the robots for the SR.
2. Max value of energy used by one of the robots for the SR.
3. Ratio of energy used on the energy at the start of the self-reconfiguration.

We made three different CNN modules, one for each of the considered energy efficiency criterion(TOTAL, MAX and RATIO). Each CNN module returns two scores for the respective distributed algorithms according to the given scenario represented by an RGB image. The distributed algorithm with the highest score is then considered the best algorithm for our criterion.

5 Experiments and analysis

5.1 Training step

The training and validation cycles of the CNN module are achieved using respectively 1088 scenarios and 504 scenarios. The last 504 scenarios are used for the test phase and correspond to the results exposed below.

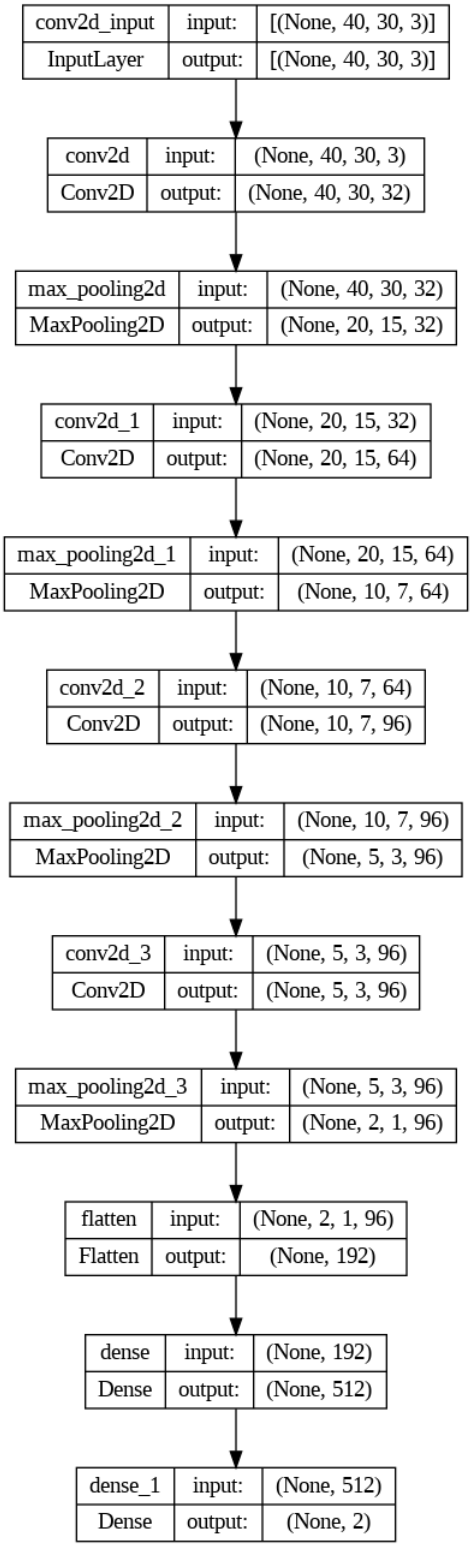


Figure 49: CNN architecture

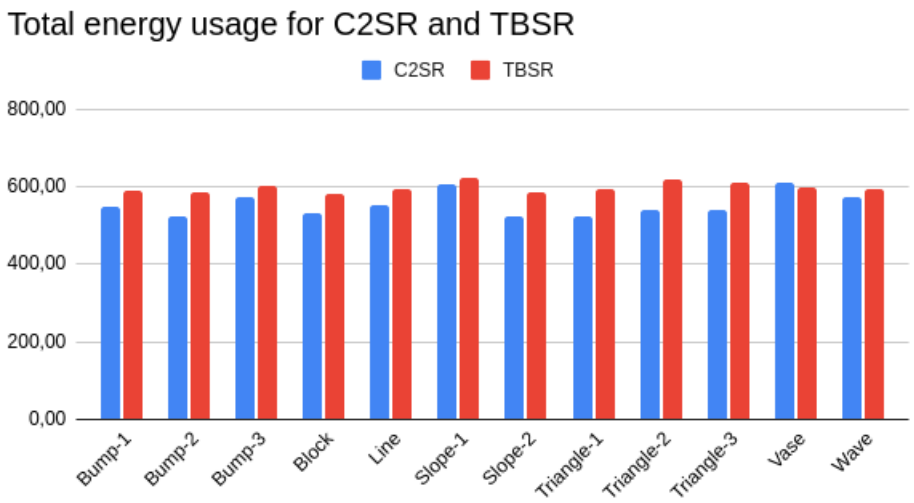


Figure 50: Total energy used for each goal shape.

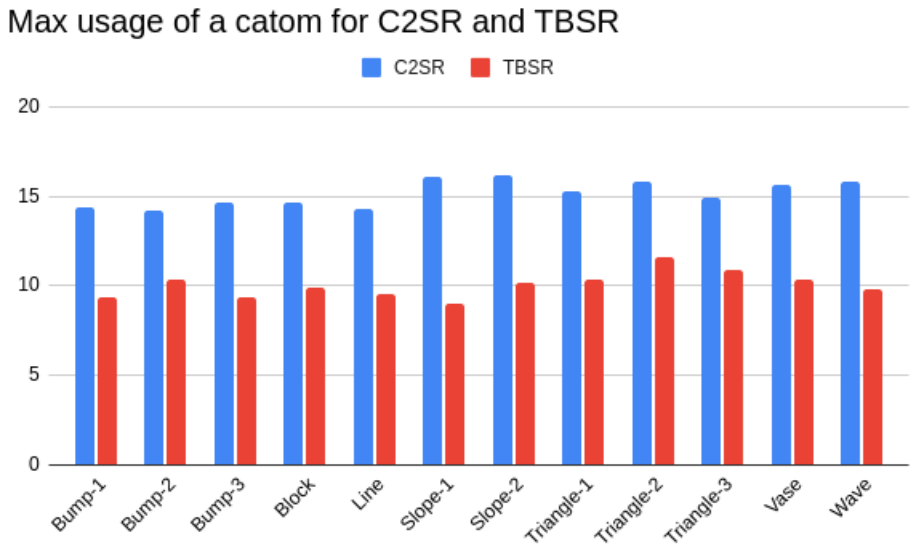


Figure 51: Max energy used by a robot for each goal shape.

5.2 Experimental results

First of all, Figures 50, 51, and 52 show that in our experiments the TBSR algorithm outperforms C2SR for all scenarios according to both the maximum energy consumption (MAX) and the maximum ratio of energy consumption criteria (RATIO). However, when evaluating based on the total energy consumption criterion (TOTAL), C2SR outperforms TBSR in the majority of instances. This outcome was expected, as TBSR was originally designed with the primary objective of ensuring equitable distribution of effort among the robots during the self-reconfiguration process.

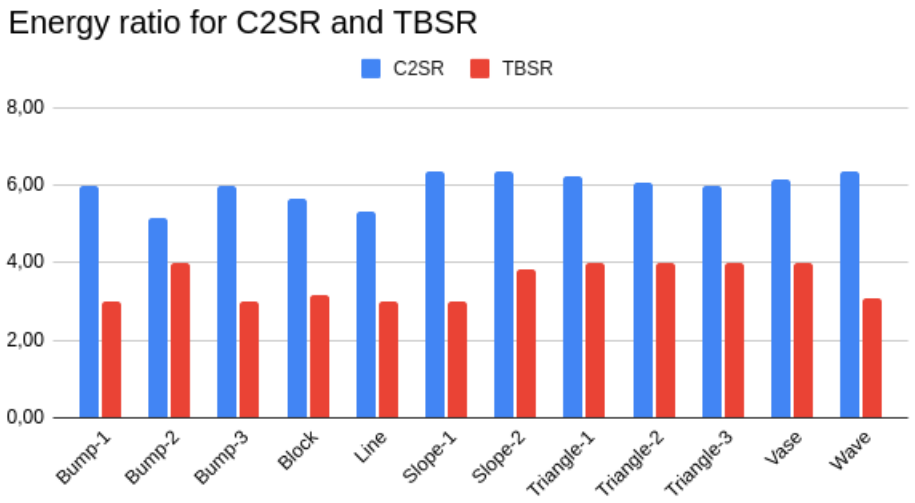


Figure 52: Ratio of energy used for each goal shape.

Bump1 to Triangle1 with template 100%		
C2SR		
TOTAL	MAX	RATIO
527,30819	17,2343	7
	TBSR	
TOTAL	MAX	RATIO
588,65	10,314	4,00

Figure 53: Result for a single case : from Bump1 to Triangle1 with template 0 (100% energy).

In figure 53, we show an example of the self-reconfiguration scenario with the initial shape set as the Bump1 shape and the goal shape set to the Triangle1 shape and all the batteries are fully charged before the self-reconfiguration. The experimental results for this example shows that TBSR improves energy efficiency according to MAX and RATIO criteria while it decreases energy efficiency regarding the TOTAL metric.
In figure 54, we display the average Total energy used, max energy used, and the ratio of energy used of TBSR and C2SR for each criterion. It show that on average, even with the biggest shapes which requires a large number of movement, the robots didn't consume a lot of energy.

5.3 Analysis

Concerning the performances of the three CNN modules in accurately identifying the most appropriate algorithm, Figure 56 displays the confusion matrices for these neural networks when evaluated on the test dataset. The Figure shows that three CNN modules achieve correct results with high accuracy, specifically 97.88% for the TOTAL criterion, and 100% accuracy for both the MAX and RATIO criteria. The training and validation curves for the accuracy and

Average for all 2096 scenarios		
C2SR		
TOTAL	MAX	RATIO
554,04	15,16	5,97
	TBSR	
TOTAL	MAX	RATIO
597,76	10,06	3,51

Figure 54: Average of results for all scenarios.

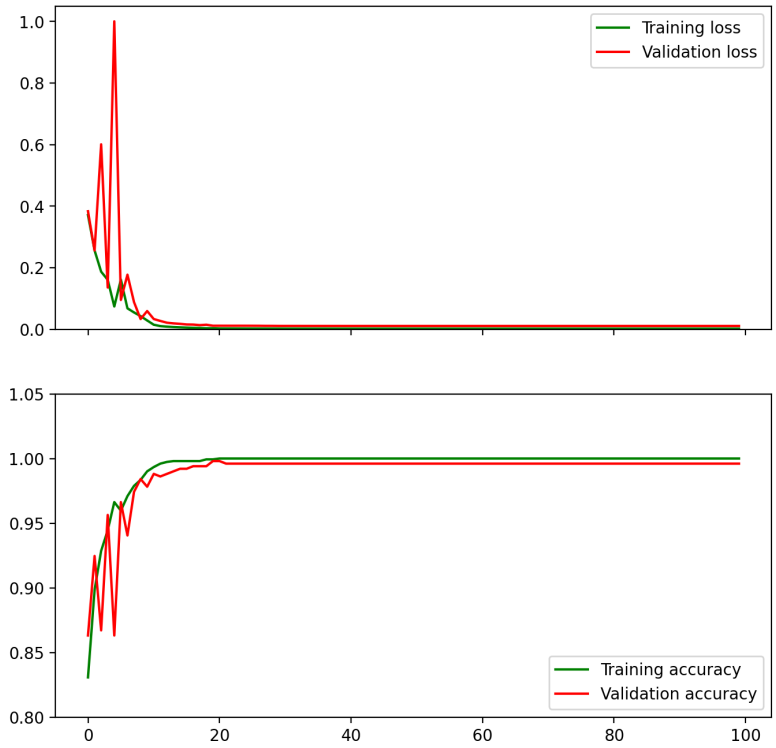


Figure 55: Training and validation metrics

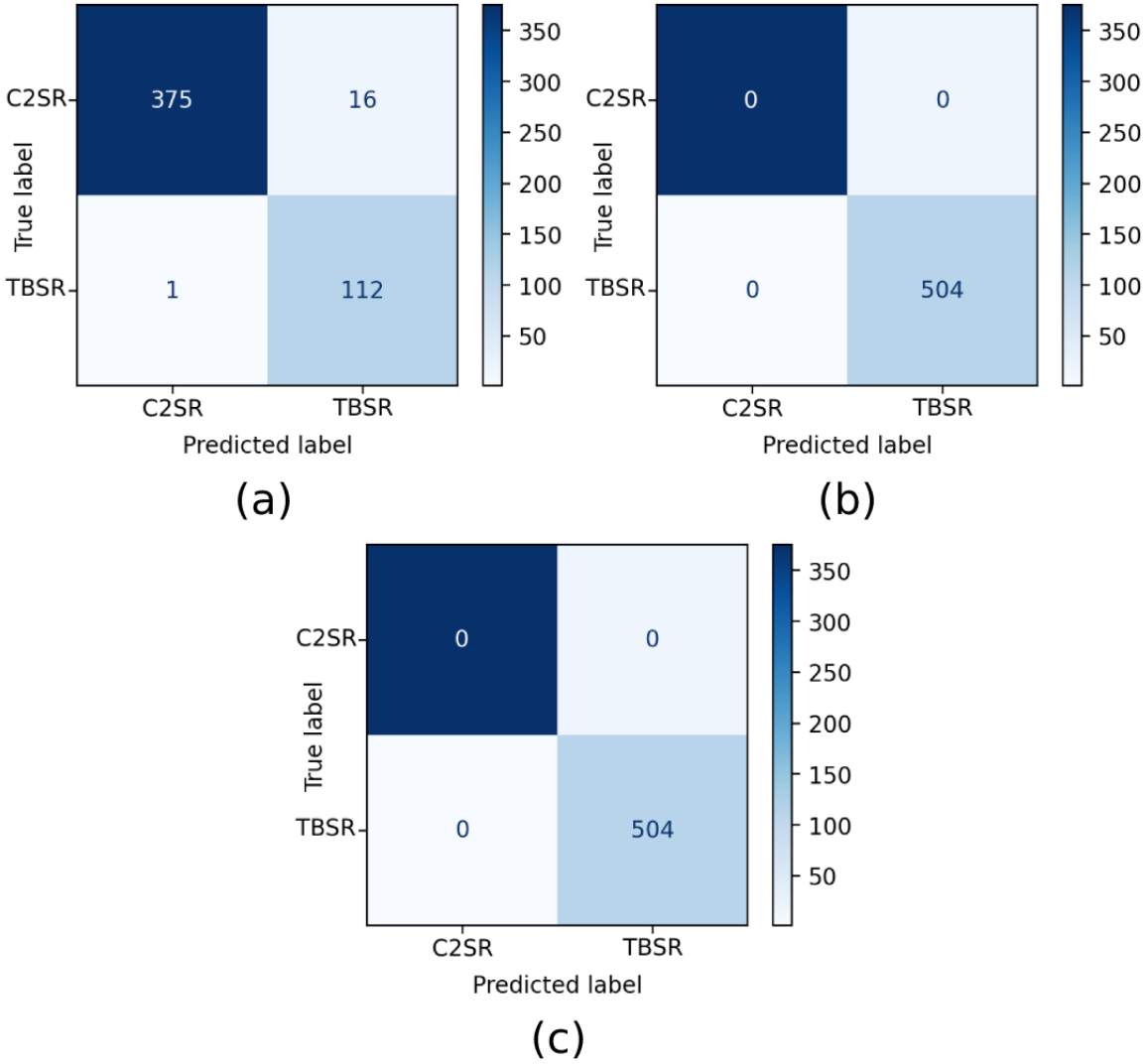


Figure 56: Confusion matrix for (a) TOTAL (b) MAX and (b) RATIO

loss, Figure 55, show that the CNN can quickly generalize the link between the data and the label.

For the same self-reconfiguration instance, the CNN3SR chosen best-suited algorithm can be different according to the considered criterion: total energy used, max energy used, and ratio of energy used. It is up to the user to choose the best energy estimation metric regarding the use case of the programmable matter. Some interesting ways to improve our approach could be the combination of multiple criteria within the same metric (by aggregation or lexicographic order) and the introduction of efficiency criteria (number of rounds for total reconfiguration).

6 Conclusion & Perspectives

While numerous challenges in achieving programmable matter have been surmounted, but the energy capacity of micro-robots to move and the network to self-reconfigurate has been largely ignored or at least crudely addressed.

In this paper, we present a first attempt at fine modeling of energy consumption of micro-robots according to the nature of the movement. This modeling is integrated into the problem of selecting a self-reconfiguration algorithm according to different criteria for evaluating energy efficiency. The resulting auto-configuration approach represents a notable advantage over existing methods in the literature. Specifically, employing an artificial neural network for identifying the optimal self-reconfiguration algorithm allows for energy cost optimization in the reconfiguration process. This optimization serves to extend the operational lifespan of both the robots and the overall system.

The results generated by CNN3SR demonstrate that the convolutional neural network can accurately identify the most suitable algorithm with an accuracy exceeding 95%. While the overall energy consumption across compared approaches may remain similar, the enhancements in maximum energy per robot and energy ratio are substantial. Consequently, robots become more proficient in replenishing their energy reserves through harvesting systems.

Chapter V
Conclusions and perspectives

In this chapter, we remind the objectives of this thesis and we emphasize the main contributions and achievements discussed above. We compare the performance discrepancies of the different approaches, taking into consideration the unique attributes of specific utilization scenarios. Following this comparative analysis, we discuss areas of improvement for the enhancement and refinement of our work.

1 Conclusions

Programmable matter as an interdisciplinary concept for computer science and robotics can help us produce new materials that have the potential to revolutionize various fields, including computer science and robotics, by offering new capabilities and opportunities. It holds great promise by offering new ways to create adaptive, versatile, and efficient systems. It has the potential to transform the way we interact with technology and design robotic systems, opening up exciting possibilities for innovation and problem-solving in a wide range of applications.

One of the primary challenges lies in the development of sophisticated control algorithms for self-reconfiguring the modular robots forming the programmable matter into different shapes. These algorithms need to enable seamless reconfiguration while considering complex factors such as module connectivity, motion planning, and stability maintenance throughout the transformation process. Achieving this requires advanced mathematical models and algorithms capable of efficiently solving complex optimization problems. The development and testing of such algorithms are time-consuming and resource-intensive endeavors.

Energy efficiency is an ever-present concern, as the size of the battery in the robot can limit us. The dynamic nature of self-reconfiguration, involving frequent movements and adjustments, can lead to increased energy consumption. Balancing the need for reconfiguration with energy conservation is vital, as it directly affects the robot's operational capabilities and longevity.

In the work presented in this thesis, we considered artificial intelligence as one of the next bricks in the development of programmable matter. In order to do so, we introduced original methods and tools to use neural networks as learning systems for self-reconfiguration of micro-robot networks. The idea is to start the reconfiguration process by an analysis of the problem instance: initial shape and final shape. The role of the Convolution Neural Network system is then to determine which distributed reconfiguration algorithm is more relevant.

To test our approach, We proposed the algorithm TBSR, a novel asynchronous distributed self-reconfiguration approach based on mass translation. TBSR shows superior performance in comparison to C2SR, with the exception of rare instances. Notably, TBSR demonstrates its efficacy by reducing the overall count of required movements by up to 17%. Furthermore, TBSR showcases its capability to equitably distribute the number of movements across the modular robots, leading to a notable reduction of up to 40% in the maximum number of movements executed by an individual robot.

With this new method, we also introduce a new way to represent data for self-reconfiguration algorithm, the Diagonally-Layered Representation (DLR), with the aim to easily describe complex shapes with only a few numbers. In addition to the fact that TBSR represents a very efficient reconfiguration algorithm, it affords two distributed self-reconfiguration algorithms (with C2SR)

to perform the CNN classification.

This original idea of applying ANN during the self-reconfiguration process of the modular robots is implemented using two CNN models called CNNSR and CNN2SR. The new self-reconfiguration approaches are hybrids between centralized and distributed methods, as they use a pre-processing step before triggering the selected distributed algorithm over the modular robots. In CNNSR approach, the CNN-based pre-processing step returns the recommended approach. In CNN2SR, the neural network system returns either the preferred algorithm or neutral if the two approaches are quite similar. The objective of CNN2SR is to improve the relevance of the returned recommendation. The Neural Network is then trained to penalize more the misclassification of TBSR-adapted instances into C2SR-adapted instances and vice versa. The misclassification of Neutral instances into C2SR to TBSR-adapted instances is less prejudicial.

Both new methods ended up giving very positive and encouraging results: CNNSR attains a success rate of 96.67% in effectively discerning the appropriate algorithm based on the characteristics of both the initial and final shapes while CNN2SR achieves a success rate of 97.25%. It is worth noting that the system can be easily extended in the future to add any desired number of self-reconfiguration algorithms.

Furthermore, we presented a third hybrid approach, called CNN3SR, that, for the first time, takes into account the initial state of charge of the modular robots' batteries. The CNN module uses an energy consumption model that estimates the required energy of each robot according to the executed self-reconfiguration algorithm. CNN3SR allows the selection of the recommended self-reconfiguration algorithm according to criteria related to energy consumption: total consumed energy over robots, maximum consumed energy by one robot, or maximum ratio of consumed energy. CNN3SR reduces also the risk of selecting an algorithm that leads to a blocking situation due to battery depletion. This way, CNN3SR improves the system fault tolerance and energy usage.

The CNN3SR system provides very encouraging results. The conducted experimentation demonstrates a commendable accuracy rate of 97% for this ANN-based algorithm selection system. In order to implement this tool, we also introduce an original model to approximate energy consumption for modular robots based on the studies on the first prototypes of 2D-Catoms and 3D-Catoms. In this context, the amount of energy consumed by a robot for a given move is computed according to the class of move performed by the robot (sliding, climbing, etc.) and the configuration of the surrounding robots.

Globally, the idea of managing the self-reconfiguration modular robot instance as an input image for a Convolutional Neural Network is a relevant and very efficient approach for combining the different distributed self-reconfiguration algorithms presented in the literature. The use of such a pre-processing step does not generate significant delay or data transmission. The CNN-based system needs just to send (broadcast) the identity of the selected algorithm over the modular robots.

2 Perspectives

Drawing from the previously discussed contributions, we have shown that artificial intelligence can be used in order to build a learning system for self-reconfiguration of micro-robot networks that will improve their performances. During the development of our tools, we have identified potential avenues for enhancement and common challenges that need to be overcome. A new

version of the system with the possibility that the CNN-based system notifies the robots of the change of distributed algorithm during the self-reconfiguration step. It is also possible to use different distributed algorithms over the modules according to each robot's situation. A deeper study of the prediction and the detection of deadlock risk is also one of the topics that AI-based approaches could be used.

We are also extremely interested in implementing our approaches in real life with actual robots, e.g., catoms[2]. Furthermore, this could allow us to study the effect of successive self-reconfiguration procedures on the same programmable matter (long-term vision). All our implementations of artificial intelligence are based on CNN, new versions using other neural networks and a comparison of all of them would be a great follow-up to this thesis. At last, implementing a new version for 3D self-reconfiguration will be the ultimate goal of this research.

List of Figures

List of Tables

References

[1] Femto-st project, 2d-catoms. `https://projects.femto-st.fr/programmable-matter/2d-catoms`, 2023.

[2] Programmable matter website. `https://www.programmable-matter.com/`, 2023.

[3] Mohamed Abdelkader, Samet Güler, Hassan Jaleel, and Jeff S Shamma. Aerial swarms: Recent applications and challenges. *Current Robotics Reports*, 2(3):309–320, 2021.

[4] Hardware and software for creating programmable matter.

[5] Hossein Ahmadzadeh, E. Masehian, and Masoud Asadpour. Modular robotic systems: Characteristics and applications. *Journal of Intelligent & Robotic Systems*, 81:317–357, 2016.

[6] Aamir Ahmed, Sandeep Arya, Vinay Gupta, Hidemitsu Furukawa, and Ajit Khosla. 4d printing: Fundamentals, materials, applications and challenges. *Polymer*, 228:123926, 2021.

[7] Joshua Auerbach, Deniz Aydin, Andrea Maesani, Przemyslaw Kornatowski, Titus Cieslewski, Grégoire Heitz, Pradeep Fernando, Ilya Loshchilov, Ludovic Daler, and Dario Floreano. RoboGen: Robot Generation through Artificial Evolution. In *IEEE Symposium on Artificial Life*, volume ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems of *ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference*, pages 136–137, 07 2014.

[8] Alexander Badri-Spröwitz, Soha Pouya, Stephane Bonardi, Jesse Kieboom, Rico Möckel, Aude Billard, Pierre Dillenbourg, and A.J. Ijspeert. Roombots: Reconfigurable robots for adaptive furniture. *IEEE Computational Intelligence Magazine*, 5, 09 2010.

[9] Saptarshi Bandyopadhyay, Soon-Jo Chung, and Fred Y. Hadaegh. Probabilistic swarm guidance using optimal transport. In *2014 IEEE Conference on Control Applications (CCA)*, pages 498–505, 2014.

[10] J Bateau, A Clark, K Mceachern, Elianne Schutze, and Jennifer Walter. Increasing the efficiency of distributed goal-filling algorithms for self-reconfigurable hexagonal metamorphic robots, 05 2023.

[11] Didier El Baz, Benoît Piranda, and Julien Bourgeois. A distributed algorithm for a reconfigurable modular surface. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, May 19-23, 2014*, pages 1591–1598. IEEE Computer Society, 2014.

[12] Julien Bourgeois, Benoit Piranda, André Naz, Nicolas Boillot, Hakim Mabed, Dominique Dhoutaut, Thadeu Tucci, and Hicham Lakhlef. Programmable matter as a cyber-physical conjugation. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 002942–002947, 2016.

[13] Baptiste Buchi, Hakim Mabed, Frédéric Lassabe, Jaafar Gaber, and Wahabou Abdou. Translation based self reconfiguration algorithm for 6-lattice modular robots. In *2021 20th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 49–56, 2021.

[14] Andres Castano, Wei-min Shen, and Peter Will. Conro: Towards deployable robots with inter-robots metamorphic capabilities. *Auton. Robots*, 8:309–324, 06 2000.

[15] Ahmed Amine Chafik, Jaafar Gaber, Souad Tayane, and Mohamed Ennaji. Programmable smart articulated interface. In f0332b0e-b4bb-47ad-a684-53702441d7cf editor.pdf, editor, *15th IEEE International Conference on Human System Interaction (HSI 2022)*, page 6, Melbourne, Australia, 6 2022.

[16] Chih-Jung Chiang and Gregory Chirikjian. Modular robot motion planning using similarity metrics. *Auton. Robots*, 10:91–106, 01 2001.

[17] G.S. Chirikjian. Kinematics of a metamorphic robotic system. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 449–455 vol.1, 1994.

[18] Francois Chollet et al. Keras, 2015.

[19] Thomas Joseph Collins and Wei-Min Shen. Rebots : A drag-and-drop high-performance simulator for modular and self-reconfigurable robots, 2016.

[20] Pawel Czarnul, Jerzy Proficz, Adam Krzywaniak, et al. Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments. *Scientific Programming*, 2019, 2019.

[21] Yawen Deng, Yiwen Hua, Nils Napp, and Kirstin Petersen. A compiler for scalable construction by the termes robot collective. *Robotics and Autonomous Systems*, 121:103240, 2019.

[22] Shlomi Dolev, Sergey Frenkel, Michael Rosenblit, Ram Prasadh Narayanan, and K Muni Venkateswarlu. In-vivo energy harvesting nano robots. In *2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE)*, pages 1–5, 2016.

[23] Ke-Lin Du and MNS Swamy. Recurrent neural networks. In *Neural networks and statistical learning*, pages 351–371. Springer, 2019.

[24] T. Fukuda and Y. Kawauchi. Cellular robotic system (cebot) as one of the realization of self-organizing intelligent universal manipulator. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 662–667 vol.1, 1990.

[25] Lidia Furno, Mogens Blanke, Roberto Galeazzi, and David Johan Christensen. Self-reconfiguration of modular underwater robots using an energy heuristic. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6277–6284. IEEE, 2017.

[26] Anirudha Ghosh, Abu Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De. Fundamental concepts of convolutional neural network. In *Recent Trends and Advances in Artificial Intelligence and Internet of Things*, pages 519–567. Springer, 2020.

[27] J Seth Copen Goldstein and Todd C. Mowry. Claytronics: An instance of programmable matter. *Wild and Crazy Ideas Session of ASPLOS*, 2004.

[28] J Seth Copen Goldstein and Todd C. Mowry. Claytronics: An instance of programmable matter. *Wild and Crazy Ideas Session of ASPLOS*, 2004.

[29] Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. Programmable matter. *IEEE Computer*, 38(6):99–101, 6 2005.

[30] Seth Copen Goldstein and Todd C. Mowry. Claytronics: A scalable basis for future robots. In *RoboSphere 2004*, Moffett Field, CA, 11 2004.

[31] Anna Gorbenko and Vladimir Popov. Programming for modular reconfigurable robots. *Programming and Computer Software*, 38:13–23, 01 2012.

[32] Enguang Guan, Zhuang Fu, Weixin Yan, Dongsheng Jiang, and Yanzheng Zhao. Self-reconfiguration path planning design for m-lattice robot based on genetic algorithm. In *Intelligent Robotics and Applications: 4th International Conference, ICIRA 2011, Aachen, Germany, December 6-8, 2011, Proceedings, Part II 4*, pages 505–514. Springer, 2011.

[33] Agneev Guin. Programmable matter - claytronics. In *Conference: 58th International Instrumentation Symposium*, 06 2012.

[34] Heiko Hamann. *Swarm Robotics: A Formal Approach*. Springer, 01 2018.

[35] Heiko Hamann and Heinz Wörn. Embodied computation. *Parallel Processing Letters*, 17:287–298, 09 2007.

[36] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[37] Rahul Haridas and Jyothi L. Convolutional neural networks: A comprehensive survey. *International Journal of Applied Engineering Research*, 14:780, 02 2019.

[38] Bent Oddvar Arnesen Haugaløkken, Martin Breivik Skaldebø, and Ingrid Schjølberg. Monocular vision-based gripping of objects. *Robotics and Autonomous Systems*, 131:103589, 2020.

[39] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. Science editions, 1949.

[40] A.J. Ijspeert, A. Martinoli, Aude Billard, and LucaMaria Gambardella. Collaboration through the exploitation of local interactions in autonomous collective robotics: The stick pulling experiment. *Autonomous Robots*, 11:149–171, 09 2001.

[41] Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. Accomplishing high-level tasks with modular robots. *Autonomous Robots*, 42:1337–1354, 2018.

[42] M.W. Jorgensen, E.H. Ostergaard, and H.H. Lund. Modular atron: Modules for a self-reconfigurable robot. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2:2068 – 2073 vol.2, 11 2004.

[43] Akiya Kamimura, Eiichi Yoshida, Satoshi Murata, Haruhisa Kurokawa, Kohji Tomita, and Shigeru Kokaji. A self-reconfigurable modular robot (mtran) — hardware and motion planning software —. In Hajime Asama, Tamio Arai, Toshio Fukuda, and Tsutomu Hasegawa, editors, *Distributed Autonomous Robotic Systems 5*, pages 17–26, Tokyo, 2002. Springer Japan.

[44] Olfa Kanoun, Sonia Bradai, Sabrine Khriji, Ghada Bouattour, Dhouha El Houssaini, Meriam Ben Ammar, Slim Naifar, Ayda Bouhamed, Faouzi Derbel, and Christian Viehweger. Energy-aware system design for autonomous wireless sensor nodes: A comprehensive review. *Sensors*, 21(2):548, 2021.

[45] Mustafa Emre Karagozler. *Design, Fabrication and Characterization of an Autonomous, Sub-Millimeter Scale Modular Robot*. PhD thesis, Carnegie Mellon University, USA, 2012. AAI3520118.

[46] Mustafa Emre Karagozler. *Design, fabrication and characterization of an autonomous, sub-millimeter scale modular robot*. PhD thesis, Carnegie Mellon University, 2012.

[47] Yonggang Ke, Luvena L. Ong, William M. Shih, and Peng Yin. Three-dimensional structures self-assembled from dna bricks. *Science*, 338(6111):1177–1183, 2012.

[48] Evelyn Fox Keller. Towards a science of informed matter. *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences*, 42(2):174–179, 2011. When Physics Meets Biology.

[49] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[50] 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.

[51] Vishaal Krishnan and Sonia Martínez. Distributed optimal transport for the deployment of swarms. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4583–4588, 2018.

[52] Christopher B. Kuhn, Markus Hofbauer, Goran Petrovic, and Eckehard Steinbach. Introspective failure prediction for autonomous driving using late fusion of state and camera information. *IEEE Transactions on Intelligent Transportation Systems*, 23(5):4445–4459, 2022.

[53] Hicham Lakhlef, Julien Bourgeois, Hakim Mabed, and Seth Copen Goldstein. Energy-aware parallel self-reconfiguration for chains microrobot networks. *Journal of Parallel and Distributed Computing*, 75:67–80, 2015.

[54] Hicham Lakhlef, Hakim Mabed, and Julien Bourgeois. Distributed and efficient algorithm for self-reconfiguration of mems microrobots. In *SAC 2013, 28-th ACM Symposium On Applied Computing*, pages 560 – 566, Coimbra, Portugal, 2013. Association for Computing Machinery (ACM).

[55] Tom Larkworthy and Subramanian Ramamoorthy. An efficient algorithm for self-reconfiguration planning in a modular robot. In *2010 IEEE International Conference on Robotics and Automation*, pages 5139–5146, 2010.

[56] Yuquan Leng, Cen Yu, Wei Zhang, Yang Zhang, Xu He, and Weijia Zhou. Task-oriented hierarchical control architecture for swarm robotic system. *Natural Computing*, 16, 12 2017.

[57] Jakub Lengiewicz and Paweł Hołobut. Efficient collective shape shifting and locomotion of massively-modular robotic structures. *Autonomous Robots*, 43:97–122, 2019.

[58] Chao Liu, Michael Whitzer, and Mark Yim. A distributed reconfiguration planning algorithm for modular robots. *IEEE Robotics and Automation Letters*, 4(4):4231–4238, 2019.

[59] Nestor Lopez, Yue Nuin, Elias Barba Moral, Lander Juan, Alejandro Solano, Víctor Vilches, and Risto Kojcev. gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo. *arXiv preprint arXiv:1903.06278*, 03 2019.

[60] H. Mabed and J. Bourgeois. Scalable distributed protocol for modular micro-robots network reorganization. *IEEE Internet of Things Journal*, 3(6):1070–1083, 12 2016.

[61] Hakim Mabed and Julien Bourgeois. Towards programmable material: Flexible distributed algorithm for modular robots shape-shifting. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM 2014, Besancon, France, July 8-11, 2014*, pages 408–414. IEEE, 2014.

[62] Hakim Mabed and Julien Bourgeois. Towards programmable material: Flexible distributed algorithm for modular robots shape-shifting. In *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 408–414, 2014.

[63] Hakim Mabed and Julien Bourgeois. Scalable distributed protocol for modular micro-robots network reorganization. *IEEE Internet of Things Journal*, 3(6):1070 – 1083, 2016.

[64] Aliah Majed, Hassan Harb, Abbass Nasser, and Benoit Clement. Run: a robust cluster-based planning for fast self-reconfigurable modular robotic systems. *Intelligent Service Robotics*, pages 1–11, 2023.

[65] Wil McCarthy. Programmable matter. *Nature*, 407(6804):569–569, 10 2000.

[66] Alessio Merlo, Mauro Migliardi, and Luca Caviglione. A survey on energy-aware security mechanisms. *Pervasive and Mobile Computing*, 24:77–90, 2015.

[67] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, and A. Ijspeert. Yamor and bluemove — an autonomous modular robot with bluetooth interface for exploring adaptive locomotion. In M. O. Tokhi, G. S. Virk, and M. A. Hossain, editors, *Climbing and Walking Robots*, pages 685–692, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[68] Satoshi Murata, Eiichi Yoshida, Akiya Kamimura, Haruhisa Kurokawa, Kohji Tomita, and Shigeru Kokaji. M-tran: Self-reconfigurable modular robotic system. *IEEE/ASME transactions on mechatronics*, 7(4):431–441, 2002.

[69] Kevin P. Murphy. Machine learning - a probabilistic perspective. In *Adaptive computation and machine learning series*, 2012.

[70] André Naz, Benoît Piranda, Julien Bourgeois, and Seth Copen Goldstein. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 254–263, 2016.

[71] André Naz, Benoît Piranda, Julien Bourgeois, and Seth Copen Goldstein. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 254–263, 2016.

[72] André Naz, Benoît Piranda, Thadeu Tucci, Seth Goldstein, and Julien Bourgeois. Network characterization of lattice-based modular robots with neighbor-to-neighbor communications. In *Distributed Autonomous Robotic Systems*, 11 2016.

[73] Jamal Nazzal, Ibrahim El-Emary, and Salam Najim. Multilayer perceptron neural network (mlps) for analyzing the properties of jordan oil shale. *World Applied Sciences Journal*, 5, 01 2008.

[74] Masaki Ohira, Ranajit Chatterjee, Tetsushi Kamegawa, and Fumitoshi Matsuno. Development of three-legged modular robots and demonstration of collaborative task execution. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3895–3900, 2007.

[75] Amit Pamecha, Chih-Jung Chiang, David Stein, and Gregory Chirikjian. Design and implementation of metamorphic robots. In *ASME 1996 Design Engineering Technical Conferences and Computers in Engineering Conference*, volume Volume 2A: 24th Biennial Mechanisms Conference of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 08 1996. V02AT02A013.

[76] Amit Pamecha, Imme Ebert-Uphoff, and Gregory S Chirikjian. Useful metrics for modular robot motion planning. *IEEE Transactions on Robotics and Automation*, 13(4):531–545, 1997.

[77] Michael Park, Sachin Chitta, Alex Teichman, and Mark Yim. Automatic configuration recognition methods in modular robots. *The International Journal of Robotics Research*, 27(3-4):403–421, 2008.

[78] Yimai Peng, Gordy Carichner, Yejoong Kim, Li-Yu Chen, Rémy Tribhout, Benoit Piranda, Julien Bourgeois, David Blaauw, and Dennis Sylvester. A 286nw, 103v high voltage generator and multiplexer for electrostatic actuation in programmable matter. In cf2b24b7-0569-4bc5-bbc3-24a72233f0d5 editor.pdf, editor, *IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits) (VLSI 2022)*, pages 158 – 159, Honolulu, United States, 7 2022.

[79] Benoit Piranda. Visiblesim: Your simulator for programmable matter. In *Dagstuhl Seminar 16271 (2016)*, volume 6, pages 12 – 12, Wadern, Germany, 2016.

[80] Benoit Piranda and Julien Bourgeois. Datom: A deformable modular robot for building self-reconfigurable programmable matter, 2020.

[81] Benoît Piranda and Julien Bourgeois. Designing a quasi-spherical module for a huge modular robot to create programmable matter. *Autonomous Robots*, 42, 12 2018.

[82] Benoît Piranda, Guillaume J. Laurent, Julien Bourgeois, Cédric Clévy, Sebastian Möbes, and Nadine Le Fort-Piat. A new concept of planar self-reconfigurable modular robot for conveying microparts. *Mechatronics*, 23(7):906–915, 2013. 1. Fractional Order Modeling and Control in Mechatronics 2. Design, control, and software implementation for distributed MEMS (dMEMS).

[83] Alberto Prieto, Beatriz Prieto, Eva Martinez Ortigosa, Eduardo Ros, Francisco Pelayo, Julio Ortega, and Ignacio Rojas. Neural networks: An overview of early research, current frameworks and new challenges. *Neurocomputing*, 214:242–268, 2016.

[84] J. W. Romanishin, K. Gilpin, and D. Rus. M-blocks: Momentum-driven, magnetic modular robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4288–4295, 2013.

[85] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE International Conference on Robotics and Automation*, pages 3293–3298, 2012.

[86] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *2012 IEEE International Conference on Robotics and Automation*, pages 3293–3298, 2012.

[87] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[88] Behnam Salemi, Mark Moll, and Wei-min Shen. Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3636–3641, 2006.

[89] Tony Salloom, Okyay Kaynak, and Wei He. A novel deep neural network architecture for real-time water demand forecasting. *Journal of Hydrology*, 599:126353, 2021.

[90] Tony Salloom, Okyay Kaynak, Xinbo Yu, and Wei He. Proportional integral derivative booster for neural networks-based time-series prediction: Case of water demand prediction. *Engineering Applications of Artificial Intelligence*, 108:104570, 02 2022.

[91] Tony Salloom, Xinbo Yu, Wei He, and Okyay Kaynak. Adaptive neural network control of underwater robotic manipulators tuned by a genetic algorithm. *Journal of Intelligent and Robotic Systems*, 97:657–672, 03 2020.

[92] Pratap Chandra Sen, Mahimarnab Hajra, and Mitadru Ghosh. Supervised classification algorithms in machine learning: A survey and review. In Jyotsna Kumar Mandal and Debika Bhattacharya, editors, *Emerging Technology in Modelling and Graphics*, pages 99–111, Singapore, 2020. Springer Singapore.

[93] Alexander Sproewitz, Aude Billard, Pierre Dillenbourg, and Auke Jan Ijspeert. Roombots-mechanical design of self-reconfiguring modular robots for adaptive furniture. In *2009 IEEE international conference on robotics and automation*, pages 4259–4264. IEEE, 2009.

[94] Kasper Støy and Haruhisa Kurokawa. Current topics in classic self-reconfigurable robot research, 2011.

[95] Hanxu Sun, Mingzhe Li, Jingzhou Song, and Yun Wang. Research on self-reconfiguration strategy of modular spherical robot. *International Journal of Advanced Robotic Systems*, 19(2):17298806221081665, 2022.

[96] Petras Swissler and Michael Rubenstein. Fireant: A modular robot with full-body continuous docks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6812–6817, 2018.

[97] Pierre Thalamy, Benoît Piranda, André Naz, and Julien Bourgeois. Visiblesim: A behavioral simulation framework for lattice modular robots. *Robotics and Autonomous Systems*, 147:103913, 2022.

[98] J. Camilo Vasquez Tieck, Sandro Weber, Terrence C. Stewart, Jacques Kaiser, Arne Roennau, and Rüdiger Dillmann. A spiking network classifies human semg signals and triggers finger reflexes on a robotic hand. *Robotics and Autonomous Systems*, 131:103566, 2020.

[99] Tommaso Toffoli and Norman Margolus. Programmable matter: Concepts and realization. *Physica D: Nonlinear Phenomena*, 47(1):263–272, 1991.

[100] McCulloch W.S. & Pitts W. A logical calculus of the ideas immanent in nervous activity., 1943.

[101] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 1 2021.

[102] Kiwon Yeom and Bum-Jae You. Three-dimensional construction based on self-reconfigurable modular robots. *International Journal of Computational Science and Engineering*, 11:368, 01 2015.

[103] M. Yim, D. G. Duff, and K. D. Roufas. Polybot: a modular reconfigurable robot. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 514–520 vol.1, 2000.

[104] Mark Yim, Paul White, Michael Park, and Jimmy Sastra. *Modular Self-Reconfigurable Robots*, pages 5618–5631. Springer New York, New York, NY, 2009.

[105] Li Zhu. *A distributed modular self-reconfiguring robotic platform based on simplified electro-permanent magnets.* Theses, Universite Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), February 2018.

[106] Yanhe Zhu, Dongyang Bie, Xiaolu Wang, Yu Zhang, Hongzhe Jin, and Jie Zhao. A distributed and parallel control mechanism for self-reconfiguration of modular robots using l-systems and cellular automata. *Journal of Parallel and Distributed Computing*, 102:80–90, 2017.

Titre : Système d'apprentissage pour le problème d'auto-reconfiguration des réseaux de micro-robots

Mots clés : Auto-reconfiguration, Algorithmes distribués, Apprentissage, Robots modulaires, Intelligence artificielle.

Résumé : Le problème d'auto-reconfiguration des réseaux de micro-robots est l'un des défis majeurs de la robotique modulaire. Un ensemble de micro-robots reliés par des liens électromagnétiques ou mécaniques se réorganisent afin d'atteindre des formes cibles données. Le problème d'auto-reconfiguration est un problème complexe pour trois raisons. Premièrement, le nombre de configurations distinctes d'un réseau de robots modulaires est très élevé. Deuxièmement, comme les modules sont libres de se mouvoir indépendamment les uns des autres, à partir de chaque configuration il est possible d'atteindre un nombre d'autres configurations lui aussi très élevé. Troisièmement et comme conséquence du précédent point, l'espace de recherche reliant deux configurations est exponentiel ce qui empêche la détermination du planning optimal de l'auto-reconfiguration.

Nous proposons dans ce travail, dans un premier temps, une approche d'auto-reconfiguration autonome distribuée TBSR, axée sur l'optimisation des déplacements pour une meilleure répartition des tâches. En d'autres termes, il s'agit de répartir l'effort fourni par chaque robot pour atteindre la forme finale.

Dans un deuxième temps, nous proposons des approches hybrides qui tirent profit des avantages des méthodes centralisées et des méthodes distribuées. Ces approches permettent de sélectionner le meilleur algorithme distribué avant le lancement de la procédure de reconfiguration. Une gamme d'algorithmes distribués sont préalablement installés sur chaque robot modulaire. Au début de la procédure d'auto-reconfiguration, un coordinateur diffuse à l'ensemble des micro-robots, les données relatives à la forme finale à atteindre et l'algorithme distribué. Pour ce faire, nous avons déterminé les caractéristiques pertinentes des problèmes d'auto-reconfiguration permettant d'identifier l'approche algorithmique la plus adaptée.

Une étude de l'impact de chaque méthode de reconfiguration et des paramètres de performances a été menée pour établir une base de connaissances. Cette base consigne les performances des divers algorithmes en fonction de différents paramètres pour un éventail varié de scénarios de problèmes d'auto-reconfiguration.

A l'aide d'un système de classification, il est ainsi possible d'établir pour chaque méthode d'auto-reconfiguration les caractéristiques des scénarios d'auto-reconfiguration pour lesquels elle se montre efficace. Les mécanismes d'apprentissage développés IA (e.g., réseaux de neurones) sont mis en œuvre. Une première approche hybride CNNSR proposée fait appel aux réseaux de neurones artificiels pour prédire l'approche optimale pour l'auto-reconfiguration. Une approche CNN2SR (une version améliorée de CNNSR), a été introduite pour la précision et la réduction des erreurs, en affinant la classification.

Dans un troisième temps, une modélisation de la consommation énergétique, issue d'expérimentations réelles avec des robots modulaires physiques (Catom 2D) a été établie. Cela a permis de mettre en œuvre une troisième approche hybride CNN3SR axé sur l'optimisation énergétique pour les robots modulaires.

Title : Learning system for self-reconfiguration of micro-robot networks

Keywords : Self-reconfiguration, Distributed algorithms, Learning, Modular robots, Artificial intelligence.

Abstract : The problem of self-reconfiguration of micro-robot networks is one of the major challenges of modular robotics. A set of micro-robots connected by electromagnetic or mechanical links reorganize themselves in order to reach given target shapes. The self-reconfiguration problem is a complex problem for three reasons. First, the number of distinct configurations of a modular robot network is very high. Secondly, as the modules are free to move independently of each other, from each configuration it is possible to reach a very high number of other configurations. Thirdly and as a consequence of the previous point, the search space connecting two configurations is exponential which prevents the determination of the optimal schedule of the self-reconfiguration.

In this work, we propose, firstly, a distributed autonomous self-reconfiguration approach TBSR, focused on the optimization of movements for a better distribution of tasks. In other words, it involves distributing the effort made by each robot to reach the final shape.

Secondly, we propose hybrid approaches that take advantage of the advantages of centralized methods and distributed methods. These approaches make it possible to select the best distributed algorithm before launching the reconfiguration procedure. A range of distributed algorithms are pre-installed on each modular robot. At the start of the self-reconfiguration procedure, a coordinator broadcasts to all the micro-robots the data relating to the final shape to be achieved and the distributed algorithm.

To do this, we determined the relevant characteristics of self-reconfiguration problems allowing us to identify the most suitable algorithmic approach.

A study of the impact of each reconfiguration method and performance parameters was conducted to establish a knowledge base. This database records the performance of various algorithms based on different parameters for a diverse range of self-reconfiguration problem scenarios.

Using a classification system, it is thus possible to establish for each self-reconfiguration method the characteristics of the self-reconfiguration scenarios for which it is effective. The learning mechanisms developed by AI (e.g., neural networks) are implemented. A first proposed hybrid CNNSR approach uses artificial neural networks to predict the optimal approach for self-reconfiguration. A CNN2SR approach (an improved version of CNNSR), was introduced for accuracy and error reduction, by refining the classification.

Thirdly, a modeling of energy consumption, resulting from real experiments with physical modular robots (Catom 2D) was established. This made it possible to implement a third hybrid CNN3SR approach focused on energy optimization for modular robots.

UBFC
UNIVERSITE
BOURGOGNE FRANCHE-COMTE

Université Bourgogne Franche-Comté
32, avenue de l'Observatoire
25000 Besançon

FEMTO-ST INSTITUTE, headquarters
15B Avenue des Montboucons - F-25030 Besançon Cedex France
Tel: (33 3) 63 08 24 00 – e-mail: contact@femto-st.fr

FEMTO-ST — AS2M: TEMIS, 24 rue Alain Savary, F-25000 Besançon France
FEMTO-ST — DISC: UFR Sciences - Route de Gray - F-25030 Besançon cedex France
FEMTO-ST — ENERGIE: Parc Technologique, 2 Av. Jean Moulin, Rue des entrepreneurs, F-90000 Belfort France
FEMTO-ST — MEC'APPLI: 24, chemin de l'épitaphe - F-25000 Besançon France
FEMTO-ST — MN2S: 15B Avenue des Montboucons - F-25030 Besançon cedex France
FEMTO-ST — OPTIQUE: 15B Avenue des Montboucons - F-25030 Besançon cedex France
FEMTO-ST — TEMPS-FREQUENCE: 26, Chemin de l'Epitaphe - F-25030 Besançon cedex France

http://www.femto-st.fr