J. Parallel Distrib. Comput. 102 (2017) 80–90

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A distributed and parallel control mechanism for self-reconfiguration of modular robots using L-systems and cellular automata

Yanhe Zhu, Dongyang Bie, Xiaolu Wang, Yu Zhang, Hongzhe Jin*, Jie Zhao

State Key Laboratory of Robotics and System, Harbin Institute of Technology, Harbin, Heilongjiang, PR China

HIGHLIGHTS

- We propose a distributed and parallel mechanism for self-reconfiguration of modular robots.
- L-systems are introduced to the distributed self-reconfiguration for a parallel system.
- The Cellular Automata are simplified with only two rules.
- This approach is convergent to target structure, robust to failure of modules and scalable to module numbers.

ARTICLE INFO

Article history:
Received 2 December 2015
Received in revised form
29 November 2016
Accepted 30 November 2016
Available online 11 December 2016

Keywords:
Modular robots
Distributed algorithm
Parallel algorithm
Self-reconfiguration
Self-repair
L-systems
Cellular automata

ABSTRACT

For distributed self-reconfiguration of Modular Self-Reconfigurable (MSR) robots, one of the main difficulties is the contradiction between limited information of decentralized modules and well-organized global structure. This paper presents a distributed and parallel mechanism for decentralized self-reconfiguration of MSR robots. This mechanism is hybrid by combining Lindenmayer systems (L-systems) describing the topological structure as configuration target and Cellular Automata (CA) for local motion planning of individual modules. Turtle interpretations are extended to modular robotics for generating module-level predictions from global description. According to local information, independent modules make motion planning by Cellular Automata in parallel. This distributed mechanism is robust to failure of modules, scalable to varying module numbers, and convergent to predefined reconfiguration targets. Simulations and statistical results are provided for validating the proposed algorithm.

1. Introduction

Modular Self-reconfigurable (MSR) robots are built by uniformly independent modules. The key value of MSR robots lies in the ability to get new configurations or functions through self-reconfiguration. A MSR robot changes its shape by reorganizing the relative positions and connection orientations of its inner modules. MSR robots are adaptable to various environments and tasks because they can change their shapes, and constitute an improvement over inflexible robots with fixed disposition of DOFs (degree of freedoms). MSR robots have potential applications in search-and-rescue missions. They are also robust against the failure of modules, since failed modules can be replaced by spare modules.

Fukuda et al. [16] provided the basic scenario that motivated research on MSR robots. Since then, several robotic systems have been designed, including M-TRAN [24], SuperBot [37], CKBot [49], Roombots [38], ATRON [30], Replicator-Symbrion [23], Sambot [48], UBot [50], Cross-ball [28], SMORES [8], and M-block [35]. Most of these systems use a hybrid architecture. Hybrid systems are typically designed to self-reconfigure through complicated implementations that approximate simple modules [34], such as the sliding cube model (SCM) [12] or the pivoting cube model [35]. Simple modules are organized in a lattice structure, for which they are called lattice-type MSR robots. The lattice organization briefs self-reconfiguration control. Assumptions can be made about the precise position of neighboring modules, and connections among neighboring modules can be open looped. The SCM module is a theoretical model of all lattice-type MSR robots for rapid verification of control method. In this paper, we take the SCM as robotic module.

This paper focuses on a fundamental problem in MSR robots, which is the distributed self-reconfiguration process [43]. In

* Corresponding author.
E-mail addresses: yhzhu@hit.edu.cn (Y. Zhu), dongyangbie@gmail.com (D. Bie), wxlu0517@gmail.com (X. Wang), zy_hitsz@163.com (Y. Zhang), hongzhejin@hit.edu.cn (H. Jin), jiezhao@hit.edu.cn (J. Zhao).

http://dx.doi.org/10.1016/j.jpdc.2016.11.016
0743-7315/© 2016 Elsevier Inc. All rights reserved.

distributed self-reconfiguration, MSR robots converge to target configurations through the interaction of independent working modules. This problem is synthetic for numerous control-related challenges that have featured as common problems in the area. According to recent work by Ahmadzadeh [2], those challenges include: (1) sufficiently robust and fault tolerant against failure of modules, (2) scalable to configurations involving different numbers of modules, (3) able to maintain global connectivity during the reconfiguration process, and (4) capable of dealing with the limitations of modules in terms of computation, sensing accuracy, and localization.

A distributed and parallel algorithm is proposed by combining Lindenmayer systems (L-systems) [27] and Cellular Automata (CA) [47,6]. L-systems are used to provide a topological description of target shape. A simplified set of CA rules is designed for independent modules to handle local movement. The gradient attraction combines global description and local movement. All modules, independently and simultaneously, take motion planning by CA and interpret L-system symbols. Turtle interpretation [1] on L-system symbols can generate module-level predictions from global description. This prediction spreads out in the system in the gradient style. Modules climb the gradient to needed areas by using CA managing local motion. MSR robots self-reconfigure by continuously attaching new modules at the turtle moving direction.

L-systems are string rewriting machines that have long been used to describe branching structures of natural growth [27,15]. The main reason for using L-systems here is their ability to compactly describe the topological structure of branching structures with connected "limbs". Branching structures with limbs at several levels can be used in a variety of applications [4,29]. The proposed algorithm is also mainly used for self-reconfiguration with branching structure as target shapes.

A cellular automaton [47] is a model of a system of cell objects. It consists of a number of things: cells, a finite set of states for each cell, neighborhood and CA rules (also called transition rules). All cells implement the CA rules for state evolving independently in discrete time, with the state of each cell at time $t + 1$ being determined by the state of its neighborhood and itself at time t, in accordance with the CA rules. In the implementation on self-reconfiguration of modular robots, each module is an independent living cell and neighboring lattice conditions constitute the neighborhood of independent modules. Then the CA rules are designed for specifying the transition of cell states, that is, the modular movement of distributed modules.

The main contributions of this paper are summarized below:

- *The proposed mechanism is distributed and parallel.* MSR robots are a distributed and parallel system consisting of multiple independently working modules. The implementation of Cellular Automata lets all modules take motion planning and move independently and in parallel. The decentralized nature of MSR robots requires that the control mechanism be distributed. The self-reconfiguration by the proposed algorithms is distributed and modules move in parallel. The number of simultaneously moving modules in each time step is recorded.
- *The proposed mechanism is convergent, robust and scalable.* The convergence to predefined structure is still an open question in exit work on distributed self-reconfiguration. The proposed algorithm is designed convergent in theory and valuated in simulations. L-systems describe target topology without predefined positions for special modules. Target form grows by adding new modules to existed structure. Modules in grown parts contain all information for the following reconfiguration. Failed modules get disconnected from the system and lost structures can grow out from the disconnecting position. L-systems can describe complex structure through rewriting simple segments. This expanding strategy makes the algorithm scalable to module numbers.
- *The use of relative positions is practical in distributed systems for modular robots.* Instead of global position, only local positions are used for localization of decentralized modules. Modules obtain their relative positions in the global state from its connected father. There are no impractical assumptions made or complex computation involved in this localization. The method is designed in consideration of practically distributed and parallel system.

2. Related work

Some methods have been used to translate the desired configurations to a pertinent description for local modules [18,32,17,36]. These methods can be classified as global centralized and local distributed models. Rules for local modules vary in different global tasks [22,39–41]. This coupling adjustment may reduce the autonomy of MSR robots. We design a simplified set of CA rules, which is effective in both locomotion and self-reconfiguration. Some impractical assumptions are also involved in those work—for example, the localization capability of a module to detect whether it is located inside the approximated volume [13,51] or on the exterior surface [14].

The basic problem of distributed control for MSR robots is to give decentralized modules the global sense through local predictions [42]. There is no general systematic way to control the behavior of each module in order to reliably achieve the desired group behavior [46,26]. Firstly, the target configuration must be described properly. And a strategy to translate this description to local predictions for decentralized modules is strongly needed. Secondly, in decentralized robotic systems, all modules work independently and an effective local control method is the foundation for successful self-reconfiguration.

For non-geometric configurations of MSR robots, L-systems are a good formalism for their high ability to describe complex structures. And the description can generate module-level predictions about global state for distributed modules through the well known turtle interpretation. In this paper, we use L-systems to provide a topological description of the self-reconfiguration target. A construction command is designed for each symbol, such that the L-systems are a sequence of commands for constructing a creature.

This sequence of commands is based on the instruction language for a Logo-style turtle [1]. The turtle interpretation can generate module-level predictions from the global description. The rewriting process also takes place in independent modules. This interpretation strategy is important for self-repair of robot systems, which is also called fault tolerance [25].

Self-reconfiguration takes place in the development style through modules continuously moving to needed areas. According to local predictions transforming in the robotic system, individual modules move in parallel. Because of the high frequency of physical interactions from contact between neighboring modules, and the constraints arising from actuator geometry and power limitations, local movements pose a significant challenge for multi-agent control [4]. Therefore local control of decentralized modules is one of the key elements for successful distributed self-reconfiguration.

We use cellular automata to handle the distributed and parallel motion of decentralized modules. Since the distributed nature of CA and the major properties of the desired controller for MSR robots are complementary, it is potential to develop algorithms based on CA. Cellular automata were invented by Von Neumann [47] and introduced to MSR robots by Butler et al. [6]. Research on distributed control in MSR robots using CA has been productive [6,7,5]. Bojinov et al. [3,4] combined gradients and CA rules to generate global configurations where rules are based on interactions with surrounding obstacles.

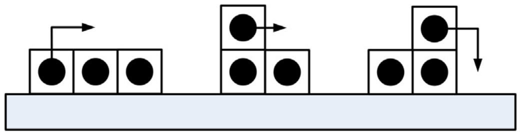


Fig. 1. The basic motion primitives of SCM. The three steps needed to move a simulated, self-reconfigurable robot that consists of three modules, represented by the squares, two steps forward.

A fundamental problem of existing work is that as tasks and configurations become more varied, the rule sets become too complex and difficult to manually design [31]. Efforts have been made to generate an automated design of CA rules, including reinforcement learning-based approaches [45] and evolutionary algorithms [31], but both methods are only effective for relatively small tasks.

An obviously simplified set of CA rules are designed in this article. Compared with work by Butler et al. concerning the use of CA for locomotion and the description of desired configurations [41,40], the CA rules used here are only intended for local motion of independent modules, and hence are much simpler with only two rules. More importantly, they are effective both for cluster-flow locomotion in obstacle environments and local motion in the self-reconfiguration process.

The rest of this paper is organized as follows: In Section 3, we outline local motion control by CA as well as the strategy for global connectivity. The basic ideas underlying L-systems and the turtle interpretation are provided in Section 4. The components of the proposed mechanism are described in Section 5. We show the dynamic self-reconfiguration of MSR robots and self-repairing property of the proposed method in Section 6. The convergence and scalability of the proposed mechanism are described in Section 7. The results of this study are summarized and ideas for future research are considered in the final section.

3. Local motion by CA and gradient

3.1. Modular motion by CA

A SCM module is a modular designed robot that can move with the ability to sense neighboring lattices, communicate with neighboring modules, and compute independently. Neighboring modules are assumed to have connection relationships. The communication of modules are limited to neighboring modules, which means that only directly neighboring modules can communicate with each other. The sensor ability of independent SCM is limited to states of directly neighboring lattices. And a module can only get information about relative position to neighboring modules.

As shown in Fig. 1, modules have two motion primitives: sliding across another module, and making a convex transition. This motion sequence can be repeated to generate cluster-flow locomotion.

A set of CA rules is designed to handle module-level motions, as shown in Fig. 2. The two rules cover all three basic motion primitives in Fig. 1. The first rule with two gradient directions covers the first and the third motion primitive of the SCM. The second rule corresponds to the second motion primitive in Fig. 1. The proposed method describes dynamic motion planning of modules, in which the gravity and rotational forces are left out. As CA rules directing 3D motion planning, modules, on the surface of robotic structure, change position independently without connection with neighboring modules. There is no condition for a module to pull others.

Compared with those complex CA rules [6,41], we only use two CA rules here. And the designed rules are effective both for locomotion in obstacle environments and for self-reconfiguration. This simplification makes full use of the modular ability to balance competitive and cooperative instincts in distributed multi-agent systems. Modular movement emerges from the interaction with neighbors.

The gradient direction in SCM modules points to one of the six connecting faces $P_m(m = 1 \cdots 6)$. A gradient direction can be in four planes $P_i(i = 1, 2, 3, 4)$ simultaneously that are all vertical to the directing face. The rules only consider local configurations in one coordinate plane P_i. The current module (called in the CA rules) checks each plane $P_i(i < 5)$ for a successful match of any CA rules.

A connecting path strategy is used to maintain global connectivity during distributed locomotion. A connecting path refers to a series of connected modules, starting from a neighboring module connected to the moving module (in CA rules) and ending at the current module. Modules in the connecting path remain connected during the motion of moving modules until receiving the free order from current modules.

All neighboring modules of the moving module must find a connecting path to the current module. Different connecting paths can have modules in common. Modules in connecting path cannot take part in any CA rules as moving modules. Thus motion of the moving module in both CA rules causes no disconnections. This strategy is effective in theory, and was verified through simulation and experiments [50].

3.2. Gradient attraction

Gradient information provides local moving direction for decentralized modules. Directly connected modules exchange gradient value through local communication on the connecting faces, as shown in Fig. 3. The initial gradient value of all modules is zero. A connecting face $P_m(m = 1 \cdots 6)$ has three kinds of neighboring lattice states:

- Module: a connected module with gradient value g', then set the gradient value $g_m = g'$ on connecting face P_m.
- Free: no module connected, the connecting face P_m has the initial gradient value $g_m = 0$.
- The turtle moving position: the neighboring lattice is on the development direction that turtles will move in (the turtle interpretation and development direction will be given in the following sections). A predefined integer (for example 20 in Fig. 3) is set to the gradient value $g_m = 20$ on connecting face P_m.

The minimum gradient value of all connecting faces $P_m(i = 1 \cdots 6)$, which must also be bigger than the inner gradient value, is reduced by one and set to the inner gradient value of this module. The gradient direction points to the connecting face with a greater gradient value than its inner gradient value.

The local communication is a key issue for computational complexity in distributed systems. Though the gradient attracting strategy has been widely used in decentralized robotic systems, modules need moving around for a while [44] or using a gradient vector [41] to obtain the local direction. A module in our improved strategy can directly obtain the gradient direction using owned gradient information, which reduces the amount of local communication, as shown in Fig. 3.

3.3. Control strategy of distributed motion

In the decentralized system, each module has a copy of the CA rules and working independently. All modules in the system keep transforming gradient information and rules matching work in parallel.

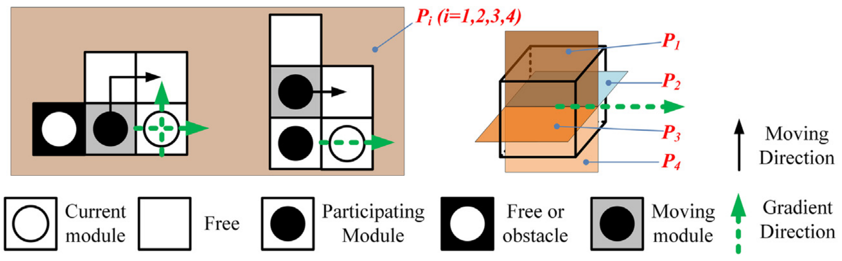


Fig. 2. A set of CA rules for local motion. Only information regarding local configuration is included, which refers to the existence of modules or obstacles in described lattice positions.

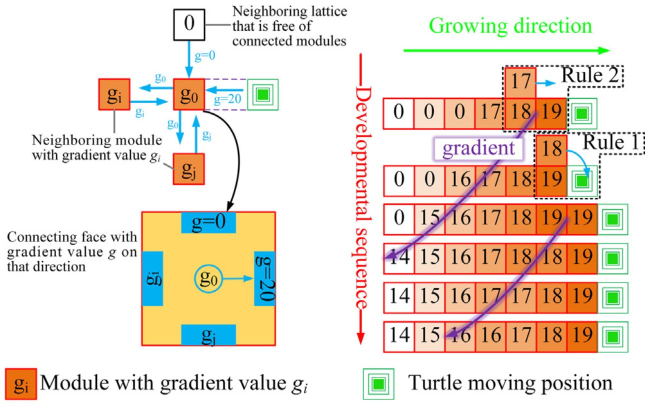

Neighboring lattice that is free of connected modules
Neighboring module with gradient value g_i
Connecting face with gradient value g on that direction
Growing direction
Developmental sequence
gradient
Rule 1
Rule 2
Module with gradient value g_i
Turtle moving position

Fig. 3. Determining local direction of motion using gradient information.

According to the gradient direction, modules, as the current module in CA rules, check the lattice conditions of relative position as rules described. The current module is also the main controller of motions in CA rules, for which it determines whether the moving module has the right to move. The right to move concerns three variables:

- *Local configuration*: The local configuration refers to the existence of modules or obstacles in those lattice positions described in the CA rules.
- *Global connectivity*: Only when all neighboring modules of the moving module find a connectivity path to the current module, can the system maintain global connectivity during distributed locomotion.
- *Configuration state*: The configuration state considers the module neither in its final state as a fixed part of self-reconfiguration results, nor in the connecting path.

When these three conditions are true, the moving module can move as CA rules description. Movements of multi modules climbing the gradient contribute to the locomotion of the whole robots. Fig. 4 shows the control strategy by CA for local motion of each module.

4. Lindenmayer systems

Lindenmayer systems (L-systems) are a mathematical theory of plant development [27]. In the distributed self-reconfiguration method, the emphasis is on configuration topology, that is the neighborhood relations among modules or groups of modules. An interpretation based on turtle geometry is used for generating module-level predictions from topology description. Basic notions related to L-system theory and turtle interpretation are provided below.

4.1. Symbol rewriting

The central concept of L-systems is symbol rewriting. Rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules or production rules. A widely used mode in graphing higher plants is node rewriting, borrowed from graph grammars [10,11]. Node rewriting appends new symbols to the original string according to rewriting rules. The rewriting process starts from an initial string of symbols, called axiom in L-systems. A simple L-system based on node rewriting is shown in Eq. (1), which has only one rewriting rule.

$$\begin{aligned}&\textit{L-system}:\\&\textit{Axiom}: \quad FX\\&\textit{Rule}: \quad X = +F - f[F] - f + FX.\end{aligned} \tag{1}$$

Starting from a simple string "*FX*" (the axiom), a sequence of strings are generated through rewriting by the production rule, as shown in Eq. (2). This extension is similar to the growth in nature. Strings can only provide a topological description of creatures, and hence the turtle interpretation is needed for their geometrical interpretation.

$$\begin{aligned}&FX\\&F + F - f[F] - f + FX\\&F + F - f[F] - f + F + F - f[F] - f + FX\\&\cdots.\end{aligned} \tag{2}$$

4.2. Turtle interpretation

The turtle interpretation [33,1] is used for graphical interpretation of L-systems. The basic idea about turtle interpretation can be found in many basic handbooks on L-systems [19].

Fig. 5 shows the corresponding graphical interpretation of the L-system in Eq. (1). In this case, both the angles δ for left '+' and right '−' turns are 90°. Starting from the axiom *FX*, the turtle moves one step to the right (drawing a line of length l with the symbol F). The system then grows depending on the reproduction rules with the command by symbol X. New expanded geometry is attached to the current position. The turtle then turns right and continues to draw a line downward, corresponding to symbols $+F$.

This rewriting strategy expands the existing part using reproduction rules. Newly appended parts arise during the rewriting of inclusion symbols. The rewriting position is at the position to which the new branches attach.

4.3. Modeling in 3D lattice environments

Configurations of lattice-based modules are 3D structures. The turtle interpretation is expanded to 3D lattice space. In lattice-based module systems, the turtle moves between connected modules. The module with turtle in, called *turtle module*, does the moving search work as the turtle. The regular organization of

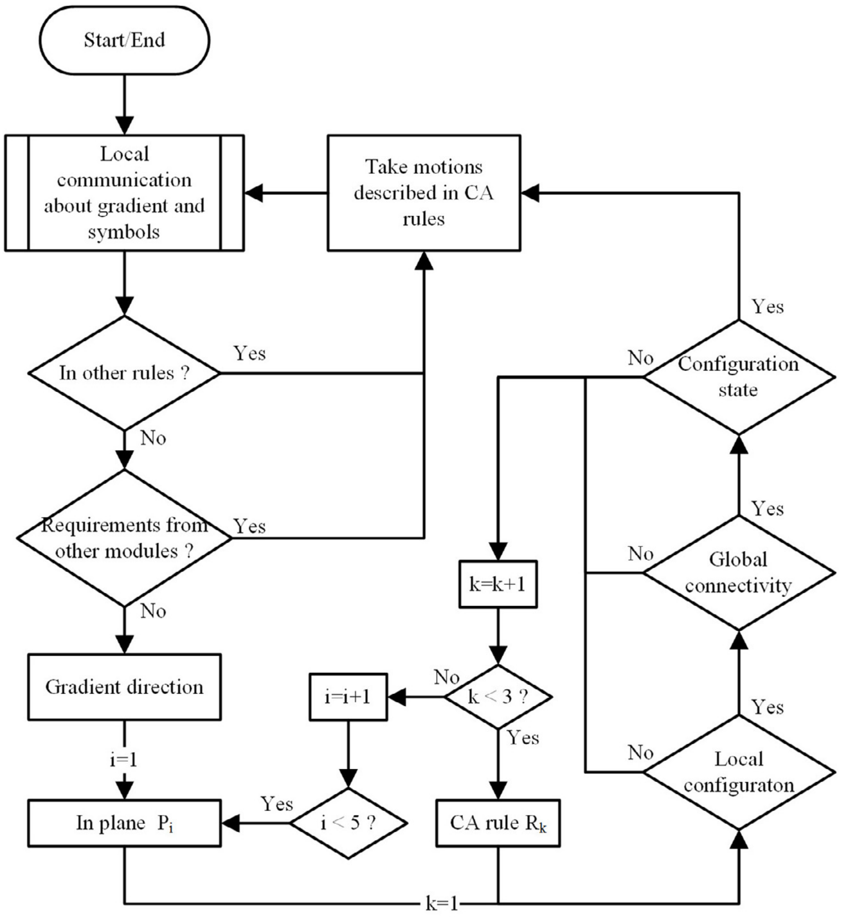


Fig. 4. The control strategy by Cellular Automata. Modules in other rules mean being Moving module or Participating module in other working rules, as shown in Fig. 2. Requirements from other modules mean in a connecting path for the global connectivity. Given a gradient direction, the current module will check all the four vertical planes (P_i, $i = 1, 2, 3, 4$, as shown in Fig. 2) that may contain a successful CA rule match.

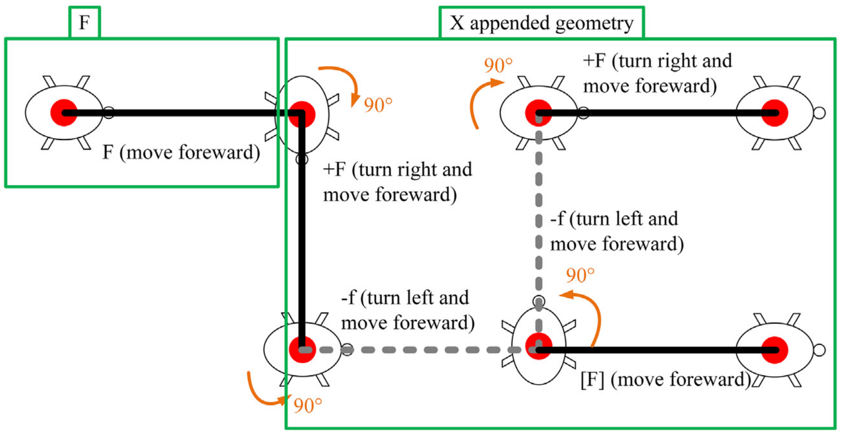

F
X appended geometry
F (move foreward)
90°
+F (turn right and move foreward)
90°
+F (turn right and move foreward)
-f (turn left and move foreward)
90°
-f (turn left and move foreward)
90°
[F] (move foreward)

Fig. 5. Turtle interpretation of a simple L-system in Eq. (1). Initially, the turtle is headed from left to right.

lattice-based modules makes it convenient for the turtle to move forward, i.e., from the connecting face which receives symbols to the opposing face.

The symbol F represents one module length. Several copies of F can appear continuously. The number of copies of F in a series represents the length of a segment. The global position is unpractical for distributed modules. In the turtle interpretation, a representation of the relative position of a module with respect to its father is used.

The character of branching structure comes from the *Push* and *Pop* operator. When a turtle module is interpreting the symbol '[', it skips and goes ahead to symbols after the corresponding ']'. A new turtle starts interpreting symbols between '[' and ']'. The segment from symbols between '[' and ']' connects to the module interpreting the symbol '['. The '[' and ']' operators can contain multi-level-nested operators for a series of branching structures. The distributed nature of MSR robotic system makes it possible of interpreting multiple turtles in parallel.

The key difficulty is to provide moving orientation for the turtle in lattice space. Orientations in 3D space are represented by three vectors, $\vec{H}$, $\vec{L}$, and $\vec{U}$, as shown in Fig. 6.

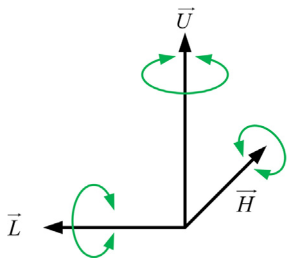


Fig. 6. The direction of the turtle in three dimensions.
Source: Image taken from [19].

These vectors, each of unit length, are perpendicular to one another and satisfy $\vec{H} \times \vec{L} = \vec{U}$. A new vector $(\vec{H}', \vec{L}', \vec{U}')$ can be obtained from vector $(\vec{H}, \vec{L}, \vec{U})$ by using rotation matrix **R**

$$(\vec{H}', \vec{L}', \vec{U}') = (\vec{H}, \vec{L}, \vec{U})\mathbf{R}. \tag{3}$$

Rotations by angle α about vectors $\vec{H}$, $\vec{L}$, and $\vec{U}$ are represented by

$$\mathbf{R}_U(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$\mathbf{R}_L(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \tag{5}$$

$$\mathbf{R}_H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}. \tag{6}$$

As in the regular organization of lattice-based modules, the rotation angle α is the integral multiple of $\pi/2$. By fixing $\alpha = \pi/2$, we obtain the matrices $\mathbf{R}_U$, $\mathbf{R}_L$, $\mathbf{R}_H$.

Since all rotations in lattice-based configurations are the integral multiple of $\pi/2$ rotations, it suffices to introduce the symbols $\mathbf{R}_U$, $\mathbf{R}_L$ and $\mathbf{R}_H$. In a similar manner to symbol F, rotation is applied as many times as the number of copies. In this way, we can represent rotations of $\pi/2$, π, and $3\pi/2$ by considering one, two, and three copies of the symbol, respectively.

5. Distributed controller

The proposed mechanism is absolutely distributed by combining L-systems and CA, as shown in Fig. 7. L-systems generate module-level predictions about global state through turtle interpretation. Turtle modules generate gradient as the gradient source to attract modules for the development of global configuration. Gradient transforms in decentralized robots through local communication and provides moving direction for distributed modules. Distributed modules take motion planning by CA rules and move along the gradient direction to needed areas.

The turtle module, a module with turtle in, does the moving search work as the turtle according to L-system descriptions. A turtle module becomes a *finalized module* when all its neighboring relationships, described in the L-system, are satisfied. *Finalized modules* will keep the relative connecting relationship as a fixed part of self-reconfiguration result. The rest of modules in the system are *spare modules*, that can climb gradient to needed areas for the development of target structure. A *spare module* changes to be a *turtle module* when connecting to a turtle module at the neighboring lattice where the turtle will move in. All modules, independently and simultaneously, spread gradient information and take motion planning by CA rules as the current module. But only spare modules can move as the described motion as moving modules.

The gradient attraction combines local motion by CA and global description by L-systems. If the neighboring lattice at moving direction is free of connected modules, the turtle module will attract spare modules to fill in the free lattice by sponsoring a gradient. In this case, the turtle module, in the aspect of interpretation, also plays as a gradient source module in the aspect of distributed locomotion by CA. For the decentralized nature of modular robots, multi turtle modules and gradient source modules are allowed to exist in parallel. Using CA rules handling local motion, spare modules climb the gradient to fill in the free lattice. The target configuration rises by attaching new modules at the developing direction.

The self-reconfiguration starts when a random module receives a string of symbols as the axiom. This module is the first turtle module in the system, from which the desired configuration grows out. Fig. 8 shows a self-reconfiguration process driven by the simple L-system in Eq. (7). All modules are initially connected in a random configuration with spare module state. Each module stores a copy of CA rules and L-system production rules. During the interpretation, the interpreted symbol is deleted and left symbols will be translated to the next turtle module. The rewriting process takes place in any turtle module interpreting the symbol 'X', which will be replaced by the string of symbols on the right of production rules.

$$\begin{array}{ll} \textit{L-system}: & \\ \textit{Axiom}: & F^6X \\ \textit{Rule}: & X = [\mathbf{R}_L F^6]F^5. \end{array} \tag{7}$$

6. Simulations

Our simulating environment is built on Microsoft Robotics Developer Studio [21] and programmed in Visual Studio using C#. The Decentralized Software Services (DSS) character has a hight match with distributed requirements of MSR robots and the proposed control mechanism.

MSR robots using the proposed mechanism can change motion styles continuously. This dynamic adjustment benefits the robots' adaptation to unknown environments. This mechanism is also self-repairing and robust to failure of modules.

6.1. Dynamic self-reconfiguration

MSR robots have two types of Self-reconfigurations: reconfiguration for changing shape or morphology, and reconfiguration for robotic movement. The second type is also called cluster-flow locomotion. Compared with traditional robots, the cluster-flow locomotion is a novel motion pattern, especially for obstacle terrains. In this article, the dynamic self-reconfiguration refers to continuous transformation between those two types of self-reconfigurations. The designed CA rules are effective in both types.

The cluster-flow locomotion is unique to MSR robots. Compared with the complex trajectory planning of quadruped walking robots, this locomotion is an optimal method for movement over terrain with unknown obstacles. As shown in Eq. (8), a L-system contains the symbol f, which commands the turtle to move forward a step length without drawing a line, as shown in Fig. 5. Correspondingly in MSR robots, finalized modules for f^l can still move to other places as spare modules. Thus the symbol f^l commands the robot to move forward a distance l in cluster-flow locomotion.

MSR robots can move through unknown environments using modules with limited computation and sensing ability. The designed CA rules can help robots adapt to obstacle environments.

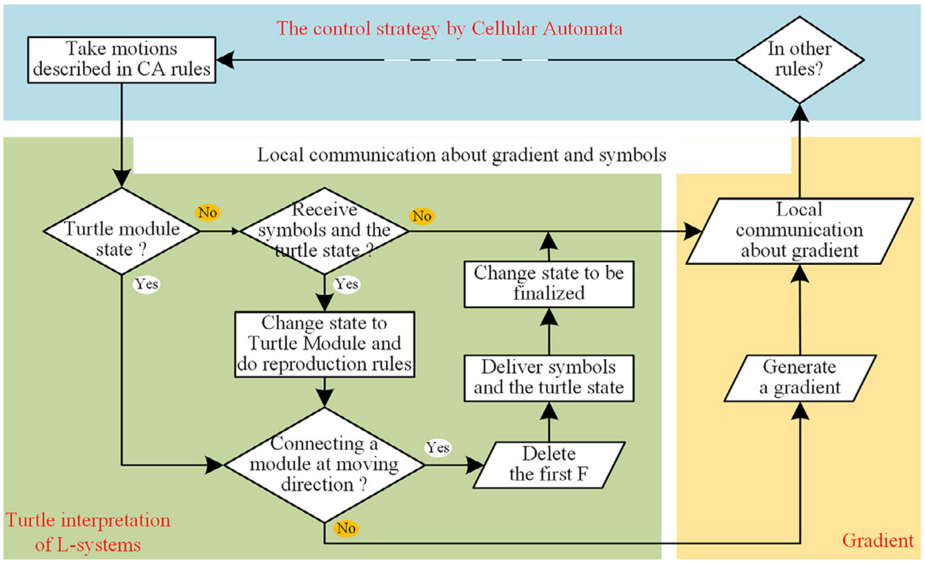

The control strategy by Cellular Automata
Take motions described in CA rules
In other rules?
Local communication about gradient and symbols
Turtle module state ?
No
Receive symbols and the turtle state ?
No
Local communication about gradient
Yes
Yes
Change state to be finalized
Change state to Turtle Module and do reproduction rules
Deliver symbols and the turtle state
Generate a gradient
Connecting a module at moving direction ?
Yes
Delete the first F
No
Turtle interpretation of L-systems
Gradient

Fig. 7. The distributed control model by combining L-systems to represent the desired configuration and CA for distributed locomotion. The "Turtle interpretation of L-systems" and "Gradient" construct the "Local communication about gradient and symbols" in the control strategy by CA shown in Fig. 4, which is given in a simplified expression here.

Modules take motion planning in real time according to local lattice situations. Obstructions are handled in decentralized modules. As shown in Fig. 9, the robot system can traverse uneven ground with lattice obstacles. The height of each lattice is generated randomly.

$$\begin{aligned} &\textit{L-system}: \\ &\quad \textit{Axiom}: \quad f(35)X \\ &\quad \textit{Rule}: \quad X = [\mathbf{R}_U F^3 \mathbf{R}_H^3 F^4][\mathbf{R}_U^3 F^3 \mathbf{R}_H F^4]F^4 X. \end{aligned} \tag{8}$$

The robot transforms from cluster-flow locomotion to shape changing reconfiguration automatically when interpreting the symbol X in Eq. (8). After getting over the uneven terrain, the robot system reconfigures to a multi-legged shape on flat topography, as shown in Fig. 9. The dynamic self-reconfiguration from locomotion to the branching structures is a novelty in the proposed algorithm.

The multi-motion style and adjustable topology are very important for real-world scenarios in search-and-rescue missions. Such as the rescue mission in earthquake relief, robots can get through obstacle terrains by using cluster-flow locomotion and then construct tree-structured agents for needed functions. Both the cluster-flow locomotion and self-reconfiguration to new structures rely on local motion of independent modules along surface of existing modules.

6.2. Self-repair

MSR robots are designed to be robust against failures of independent or groups of modules. The proposed control method corresponds to the nature of robustness. This character leads to another important feature, self-repair. The lost or failed parts can be replaced with spare modules. As shown in Fig. 10, due to internal issues or external impact, a few modules are disconnected from the global structure. A group of modules move to the system and climb it to repair the lost parts.

This self-repair character requires modules to record containing L-system symbols when they change to finalized module from turtle module. When a finalized module finds that its local configuration cannot satisfy the description of the symbols, it turns into a turtle module and restarts the turtle interpretation. This is what happened to the module at the breaking position in Fig. 10. Spare modules move on the surface and replace the lost part. The robot system turns out to be self-repairing and robust against modular failures.

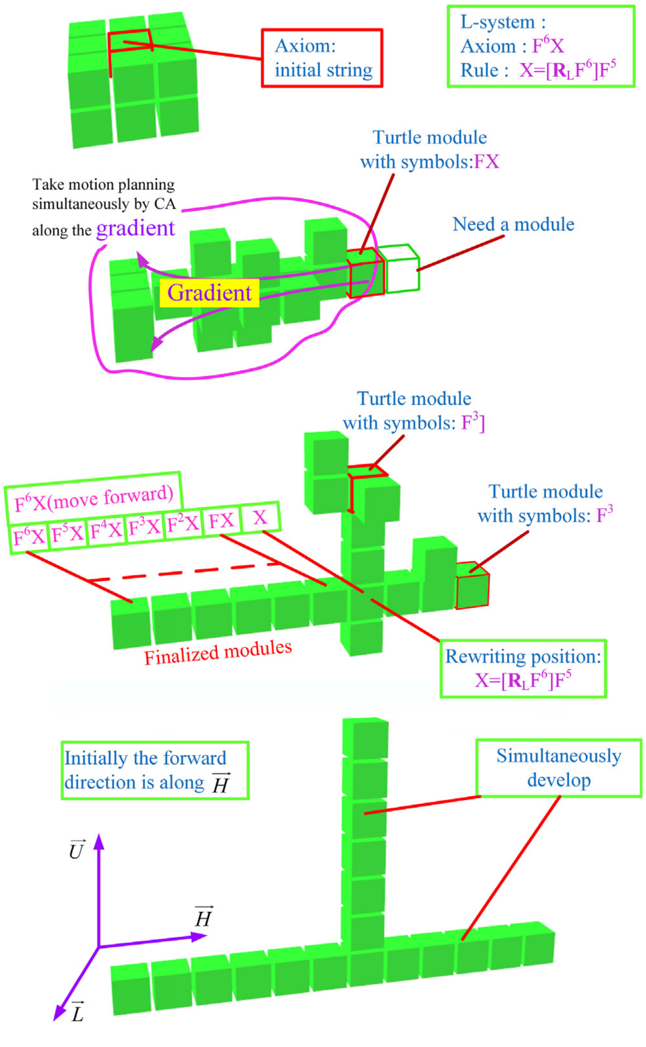

Axiom: initial string
L-system : Axiom : F^6X Rule : $X=[R_LF^6]F^5$
Turtle module with symbols:FX
Take motion planning simultaneously by CA along the gradient
Need a module
Gradient
Turtle module with symbols: F^3]
F^6X(move forward)
F^6X F^5X F^4X F^3X F^2X FX X
Turtle module with symbols: F^3
Finalized modules
Rewriting position: $X=[R_LF^6]F^5$
Initially the forward direction is along $\vec{H}$
Simultaneously develop
$\vec{U}$
$\vec{H}$
$\vec{L}$

Fig. 8. The self-reconfiguration process with the L-system in Eq. (7).

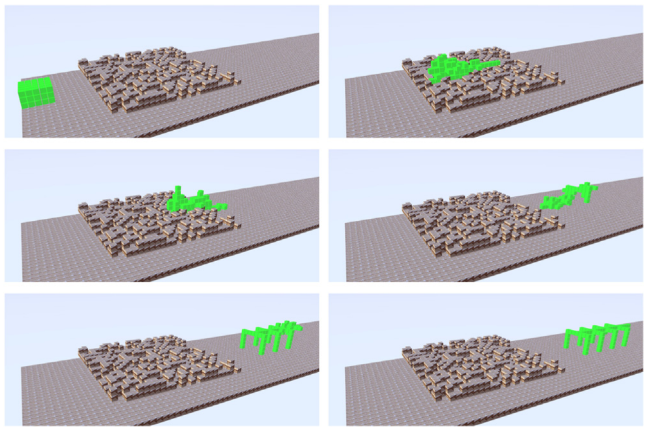


Fig. 9. The dynamic self-reconfiguration of MSR robots.

7. Discussion

The proposed mechanism is convergent to predefined topology and scalable to module numbers. Multiple simulations and statistical analysis are provided.

7.1. Convergence to predefined structure

In order to illustrate the convergence of the proposed algorithm, various self-reconfigurations are given with global structures generated using sequential reproduction of L-systems. Both experiments in Figs. 11 and 13 are repeated 50 times. The total time for convergence in each experiment is recorded. A simple L-system is designed in Eq. (9) and the corresponding self-reconfiguration with 75 modules is shown in Fig. 11. The number of simultaneously moving modules at each time step is recorded.

$$\begin{array}{ll} \textit{L-system}: & \\ \textit{Axiom}: & F^{11}X \\ \textit{Rule}: & X = \mathbf{R}_L^3F^8[\mathbf{R}_UF^8\mathbf{R}_H^3F^{10}] \\ & \quad [\mathbf{R}_U^3F^8\mathbf{R}_HF^{10}]F^8\mathbf{R}_L^3F^{10}. \end{array} \tag{9}$$

The convergence state is calculated by the ratio of n_1 to n_2, where n_1 stands for the number of finalized modules as a fixed part of reconfiguration result. And n_2 stands for the number of total modules needed by the desired configuration. The system completes self-reconfiguration when the ratio increases to 100%. Systems in all 50 experiments converge to the desired configuration in a limited number of time steps. Average results are shown in Fig. 12.

Parallel movements of modules conform to the parallel and distributed character of proposed algorithm. The number of simultaneously moving modules increases rapidly at the beginning of self-reconfiguration. This increase comes from the spread of gradient. More and more modules start motion planning by CA rules when sensing the gradient. Along with the self-reconfiguration, modules become their finalized state as a fixed part of final configuration. Then the decrease of simultaneously moving modules comes from reduced number of modules that can move. As shown in Fig. 12, there are no more than 10 modules moving in parallel after 200 steps, with the convergence rate 91%.

Another experiment is reconfiguring to a 3D snow in Fig. 13. The generation of snow is commonly used in demonstrating L-systems and computer graphics. We extend it to 3D lattice surroundings.

More than 90% self-reconfiguration completes after 200 steps in Fig. 12 and 500 steps in Fig. 13. But the rest 10% self-reconfiguration in both experiments requires a relatively large number of time steps. This phenomenon is due to the branching character of global

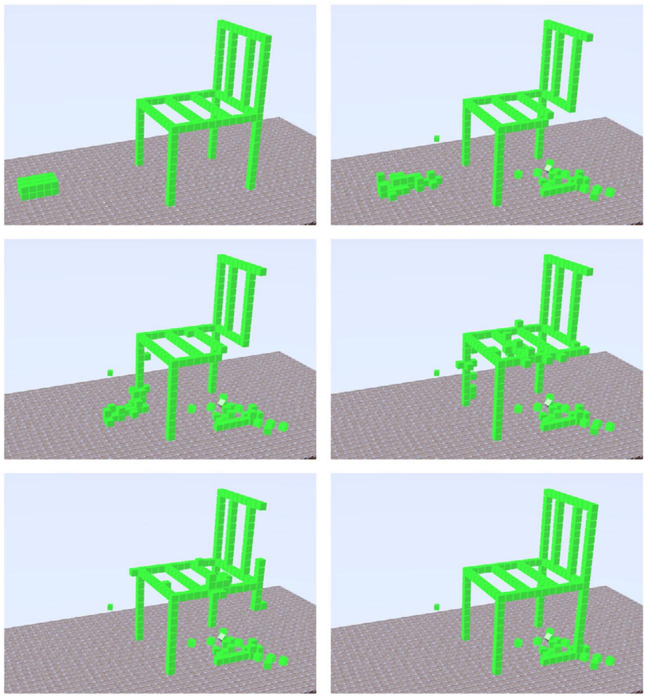


Fig. 10. Robustness of MSR robot systems against module failure, and their self-repair property.

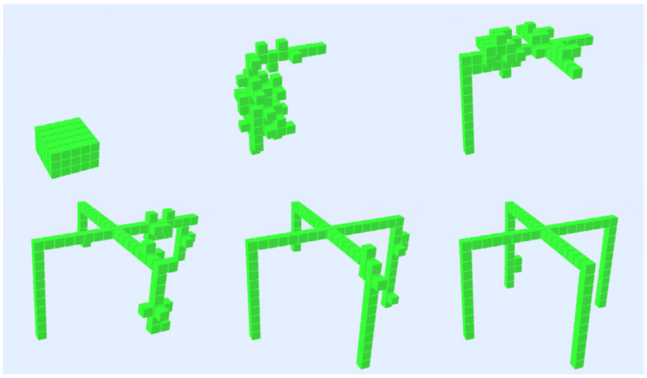


Fig. 11. The self-reconfiguration process for a branching structure.

structure. There are spare modules at the end of some finished segments. They need to move to the unfinished limbs. Both the spread of the gradient information from the unfinished segments to the spare modules and the movement of spare modules after receiving gradient sense require some time steps. As shown in Fig. 12 (right), fewer than eight modules keep moving after 200 steps.

The convergence of all experiments in Figs. 11 and 13 supports the claim that the proposed algorithm is convergent to branching structures.

7.2. Scalability of module numbers

Another important character of MSR systems is scalable to module numbers. A MSR robot may have varying numbers of modules for different tasks. The scalability of modular design requires the scalability of control methods. A series of simulations are provided to demonstrate the scalability of the proposed method.

The rewriting strategy in L-systems contributes to scalability with self-similar self-configuration results. Rewriting can define

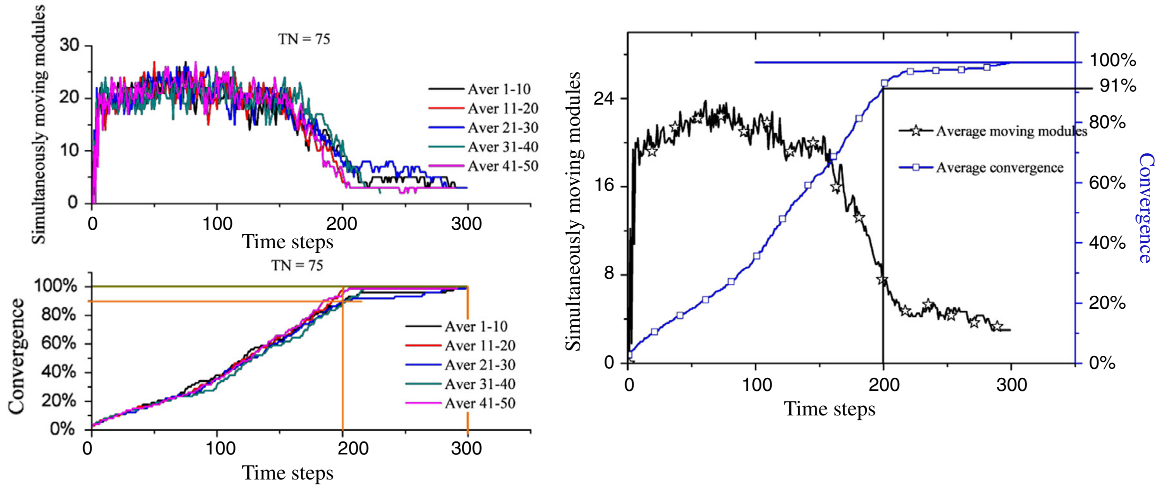


Fig. 12. Statistical results of the experiment in Fig. 11. The left two figures show simultaneously moving modules at each time step and the converging process as time goes on. The right figure shows the interaction between convergence state and simultaneously moving modules.

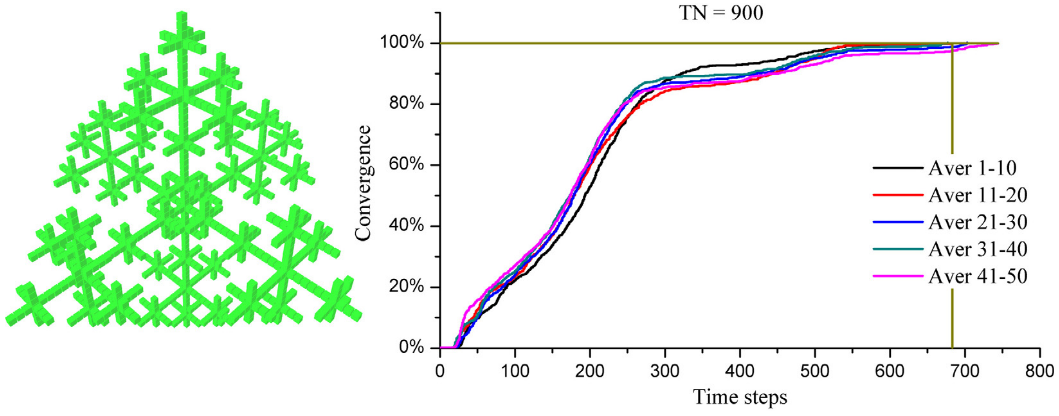


Fig. 13. An experiment of self-reconfiguration to a 3D snow and the statistical result about convergence. The snow configuration is a typical use of rewriting strategy by repeating similar structures at different positions. It will be a complex work for manually designing the global structure.

complex objects by successively repeating simple initial objects, as shown in the L-system in Eq. (10). During self-reconfiguration, new parts generate by repeating the basic structure described in reproduction rules.

$$
\begin{aligned}
&\textit{L-system}: \\
&\textit{Axiom}: \quad F^{11}X \\
&\textit{Rule}: \quad X = \mathbf{R}_L^3F^8[\mathbf{R}_UF^8\mathbf{R}_H^3F^{10}] \\
&\qquad\qquad [\mathbf{R}_U^3F^8\mathbf{R}_HF^{10}]X.
\end{aligned} \tag{10}
$$

We repeat the experiments in Eq. (10) and Fig. 13 with an increasing number of modules. Each system with a fixed *TN* is repeated 50 times. We record the total time and the corresponding global state during self-reconfiguration. In all 400 experiments, the system converges to the desired configuration in a limited number of time steps. As shown in Fig. 14, the self-reconfiguration results are self-similar by repeating the basic structure described by *X* in Eq. (10). The result is another support for the claim that the proposed control method is convergent and scalable to the number of modules.

Statistical results in both Figs. 14 and 15 show how time steps for convergence increase as the number of modules. The time step for each self-reconfiguration is the average value of 50 experiments with the same *TN*. The number of time steps for self-reconfiguration grows linearly with the number of modules. This linear increase is a great forward relative to the exponential growth in optimal reconfiguration planning [20].

8. Results and future work

A control mechanism is proposed for distributed self-reconfiguration targeting at branching structures. The mechanism is distributed by combining L-systems and CA. L-systems are taken to present topological structure and provide independent modules with global sense through module-level predictions. A simplified set of CA rules is designed to handle local motion planning. The proposed mechanism is efficient, robust, scalable, and convergent, which are verified through various simulations and statistical results.

The proposed mechanism is an absolutely distributed control strategy. While CA shares the decentralized nature with modular robots, the introduction of L-systems affects a transformation of symbols between connected modules. The turtle interpretation is extended to generate module-level information regarding the global state. This interpretation of symbols uses relative positions instead of global positions, which fits well with the distributed nature of modular robots. Gradient information is also translated through local communication between connected modules.

Robot systems using the proposed mechanism can directly reconfigure to desired structures from initial state. This solves requirements for the computation of reconfiguration feasibility between initial and final configurations [9]. Self-repairing is another important feature of the proposed mechanism, whereby disconnected parts can be replaced by spare modules.

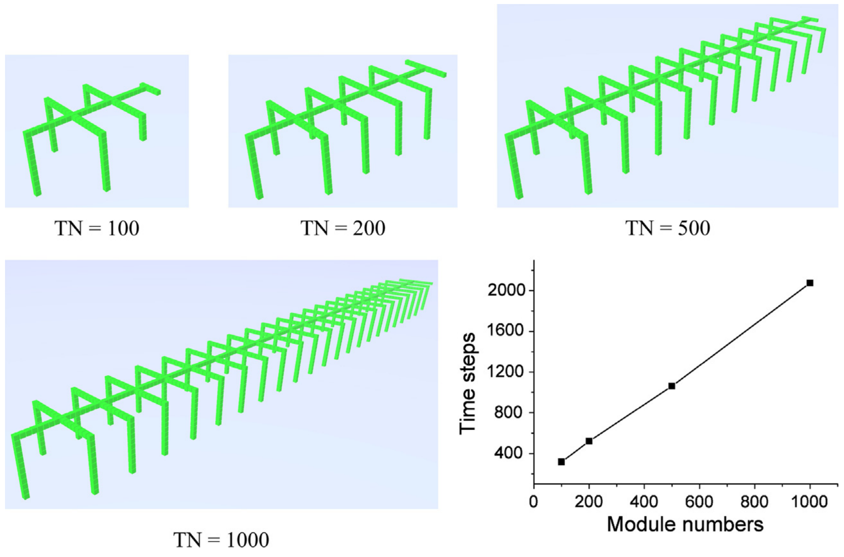


Fig. 14. Statistical results about scalability to module numbers of self-reconfiguration by Eq. (9).

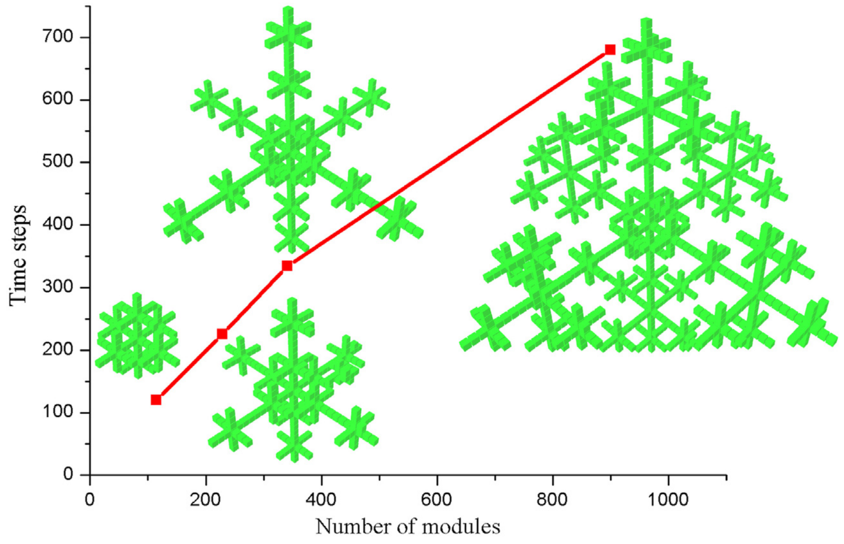


Fig. 15. The scalability to module numbers of self-reconfiguration in Fig. 13.

There remain problems that require further research. Although the relative position in the turtle interpretation is practical for mechanical modules in distributed systems, the reliability of local communication between physically realized modules remains a challenging issue. This will be the focus of our future work.

Another important problem is the manually designed reproduction rules of L-systems. New strategies to automatically generate the axiom and the reproduction rules are sorely needed. In addition to the automatic algorithm, the influence of environment and the objectives should also be considered. The interaction between reconfiguration process and outside stimuli may lead to the automatic emergence of functional configurations.

Acknowledgment

This work was supported by National Natural Science Foundation of China (Grant No. 61273316 & 61673137).

Appendix A. Supplementary data

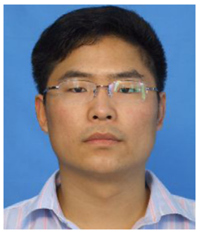
Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.jpdc.2016.11.016.

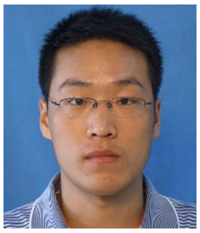
References

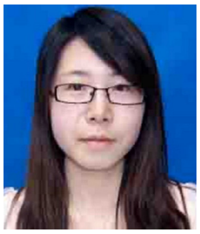
[1] H. Abelson, Turtle Geometry: The Computer as a Medium for Exploring Mathematics, MIT Press, 1986.
[2] H. Ahmadzadeh, E. Masehian, Modular robotic systems: Methods and algorithms for abstraction, planning, control, and synchronization, Artificial Intelligence 223 (2015) 27–64.
[3] H. Bojinov, A. Casal, T. Hogg, Emergent structures in modular self-reconfigurable robots, in: Robotics and Automation, Vol. 2, IEEE, 2000, pp. 1734–1741.
[4] H. Bojinov, A. Casal, T. Hogg, Multiagent control of self-reconfigurable robots, Artificial Intelligence 142 (2) (2002) 99–120.
[5] Z. Butler, K. Kotay, D. Rus, Generic decentralized control for lattice-based self-reconfigurable robots, Int. J. Robot. Res. 23 (9) (2004) 919–937.
[6] Z. Butler, K. Kotay, D. Rus, K. Tomita, Cellular automata for decentralized control of self-reconfigurable robots, in: Proc. IEEE ICRA Workshop on Modular Robots, Citeseer, 2001.
[7] Z. Butler, D. Rus, Distributed locomotion algorithms for self-reconfigurable robots operating on rough terrain, in: 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2003. Proceedings, Vol. 2, IEEE, 2003, pp. 880–885.
[8] J. Davey, N. Kwok, M. Yim, Emulating self-reconfigurable robots-design of the SMORES system, in: Intelligent Robots and Systems, (IROS), IEEE, 2012, pp. 4464–4469.
[9] A. Dumitrescu, I. Suzuki, M. Yamashita, Motion planning for metamorphic systems: Feasibility, decidability, and distributed reconfiguration, IEEE Trans. Robot. Autom. 20 (3) (2004) 409–418.

[10] R. Erickson, The geometry of phyllotaxis, in: The Growth and Functioning of Leaves, 1983, pp. 53–88.
[11] G. Eyrolles, Synthèse d'images figuratives d'arbres par des méthodes combinatoires (Ph.D. thesis), 1986.
[12] R. Fitch, Z. Butler, D. Rus, Reconfiguration planning for heterogeneous self-reconfiguring robots, in: Intelligent Robots and Systems, 2003, Vol. 3, IROS 2003, IEEE, 2003, pp. 2460–2467.
[13] R. Fitch, Z. Butler, D. Rus, Reconfiguration planning among obstacles for heterogeneous self-reconfiguring robots, in: Robotics and Automation, 2005, ICRA 2005, 2005, pp. 117–124. http://dx.doi.org/10.1109/ROBOT.2005.1570106.
[14] R. Fitch, R. McAllister, Hierarchical planning for self-reconfiguring robots using module kinematics, in: Distributed Autonomous Robotic Systems, Springer, 2013, pp. 477–490.
[15] D. Frijters, A. Lindenmayer, Developmental descriptions of branching patterns with paracladial relationships, Automata, languages, development, 1976, 57–73.
[16] T. Fukuda, S. Nakagawa, Dynamically reconfigurable robotic system, in: Robotics and Automation, IEEE, 1988, pp. 1581–1586.
[17] S. Funiak, P. Pillai, M.P. Ashley-Rollman, J.D. Campbell, S.C. Goldstein, Distributed localization of modular robot ensembles, Int. J. Robot. Res. 28 (8) (2009) 946–961.
[18] S.C. Goldstein, J.D. Campbell, T.C. Mowry, Programmable matter, Computer 38 (6) (2005) 99–101.
[19] J.S. Hanan, A. Lindenmayer, P. Prusinkiewicz, The Algorithmic Beauty of Plants, Springer-Verlag, 1990.
[20] F. Hou, W.-M. Shen, Graph-based optimal reconfiguration planning for self-reconfigurable robots, Robot. Auton. Syst. 62 (7) (2014) 1047–1059.
[21] J. Jackson, Microsoft robotics studio: A technical introduction, IEEE Robot. Autom. Mag. 14 (4) (2007) 82–87. http://dx.doi.org/10.1109/M-RA.2007.905745.
[22] C. Jones, M.J. Mataric, From local to global behavior in intelligent self-assembly, in: Robotics and Automation, Vol. 1, IEEE, 2003, pp. 721–726.
[23] S. Kernbach, E. Meister, O. Scholz, R. Humza, J. Liedke, L. Ricotti, J. Jemai, J. Havlik, W. Liu, Evolutionary robotics: The next-generation-platform for on-line and on-board artificial evolution, in: Evolutionary Computation, 2009, CEC', IEEE, 2009, pp. 1079–1086.
[24] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, S. Murata, Distributed self-reconfiguration of M-TRAN III modular robotic system, Int. J. Robot. Res. 27 (3–4) (2008) 373–386.
[25] H. Lakhlef, J. Bourgeois, H. Mabed, S.C. Goldstein, Energy-aware parallel self-reconfiguration for chains microrobot networks, J. Parallel Distrib. Comput. 75 (2015) 67–80.
[26] S. Li, L. Li, G. Lee, H. Zhang, A hybrid search algorithm for swarm robots searching in an unknown environment, PLoS One 9 (11) (2014) e111970.
[27] A. Lindenmayer, Mathematical models for cellular interactions in development i. Filaments with one-sided inputs, J. Theoret. Biol. 18 (3) (1968) 280–299.
[28] Y. Meng, Y. Zhang, A. Sampath, Y. Jin, B. Sendhoff, Cross-ball: a new morphogenetic self-reconfigurable modular robot, in: Robotics and Automation, (ICRA), IEEE, 2011, pp. 267–272.
[29] H. Moravec, Mind Children: The Future of Robot and Human Intelligence, Harvard University Press, 1988.
[30] E.H. Østergaard, K. Kassow, R. Beck, H.H. Lund, Design of the ATRON lattice-based self-reconfigurable robot, Auton. Robots 21 (2) (2006) 165–183.
[31] E.H. Ostergaard, H.H. Lund, Distributed cluster walk for the ATRON self-reconfigurable robot, in: the The 8th Conference on Intelligent Autonomous Systems, IAS-8, 2004, pp. 291–298.
[32] P. Pillai, J. Campbell, G. Kedia, S. Moudgal, K. Sheth, A 3D fax machine based on claytronics, in: 2006 IEEE/RSJ Intelligent Robots and Systems, IEEE, 2006, pp. 4728–4735.
[33] P. Prusinkiewicz, Graphical applications of L-systems, in: Proceedings of Graphics Interface, Vol. 86, 1986, pp. 247–253.
[34] J.W. Romanishin, K. Gilpin, S. Claici, D. Rus, 3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions, in: Robotics and Automation, (ICRA), IEEE, 2015, pp. 1925–1932.
[35] J.W. Romanishin, K. Gilpin, D. Rus, M-blocks: Momentum-driven, magnetic modular robots, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS), IEEE, 2013, pp. 4288–4295.
[36] M. Rubenstein, W.-M. Shen, Automatic scalable size selection for the shape of a distributed robotic collective, in: Intelligent Robots and Systems, (IROS), IEEE, 2010, pp. 508–513.
[37] W.-M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein, J. Venkatesh, Multimode locomotion via SuperBot reconfigurable robots, Auton. Robots 20 (2) (2006) 165–177.
[38] A. Spröwitz, R. Moeckel, M. Vespignani, S. Bonardi, A.J. Ijspeert, Roombots: A hardware perspective on 3D self-reconfiguration and locomotion with a homogeneous modular robot, Robot. Auton. Syst. 62 (7) (2014) 1016–1033.
[39] K. Stoy, Controlling self-reconfiguration using cellular automata and gradients, in: Proceedings of the 8th International Conference on Intelligent Autonomous Systems, IAS-8, 2004, pp. 693–702.
[40] K. Stoy, How to construct dense objects with self-recondfigurable robots, in: European Robotics Symposium 2006, Springer, 2006, pp. 27–37.
[41] K. Stoy, Using cellular automata and gradients to control self-reconfiguration, Robot. Auton. Syst. 54 (2) (2006) 135–141.
[42] K. Stoy, Lattice automata for control of self-reconfigurable robots, in: Robots and Lattice Automata, Springer, 2015, pp. 33–45.
[43] K. Stoy, D. Brandt, D.J. Christensen, Self-Reconfigurable Robots: An Introduction, MIT Press, 2010.
[44] K. Stoy, R. Nagpal, Self-reconfiguration using directed growth, in: Distributed Autonomous Robotic Systems, Vol. 6, Springer, 2007, pp. 3–12.
[45] P. Varshavskaya, L.P. Kaelbling, D. Rus, Automated design of adaptive controllers for modular robots using reinforcement learning, Int. J. Robot. Res. 27 (3–4) (2008) 505–526.
[46] M. Vigelius, B. Meyer, G. Pascoe, Multiscale modelling and analysis of collective decision making in swarm robotics, PLoS One 9 (11) (2014) e111542.
[47] J. Von Neumann, Theory of self-reproducing automata, IEEE Trans. Neural Netw. 5 (1) (1994) 3–14.
[48] H. Wei, Y. Chen, J. Tan, T. Wang, Sambot: A self-assembly modular robot system, IEEE/ASME Trans. Mechatronics 16 (4) (2011) 745–757.
[49] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, G.S. Chirikjian, Modular self-reconfigurable robot systems [grand challenges of robotics], IEEE Robot. Autom. Mag. 14 (1) (2007) 43–52.
[50] Y. Zhu, D. Bie, S. Iqbal, X. Wang, Y. Gao, J. Zhao, A simplified approach to realize cellular automata for UBot modular self-reconfigurable robots, J. Intell. Rob. Syst. (2014) 1–18.
[51] M. Zoppi, Self-reconfigurable robots: An introduction, Ind. Robot: Int. J. 38 (4) (2010).

Yanhe Zhu is a Professor of Robots and control with State Key Laboratory of Robotics and System, Harbin Institute of Technology, China. He is broadly interested in design and control of Modular robots, wearable skeleton robots and bio-robots.


Dongyang Bie is a Ph.D. candidate in Robotics and systems at Harbin Institute of Technolgoy, China. He is studying modular robots and soft robots under Pro. Jie Zhao and Pro. Yanhe Zhu. His research interests include distributed control of modular robots, obstacle avoidances of mobile robots, artificial intelligence of swarm agents.


Xiaolu Wang is a Ph.D. candidate in Robotics and systems at Harbin Institute of Technolgoy, China. He is studying modular robots under Pro. Jie Zhao and Pro. Yanhe Zhu. His research interests include motion control of modular robots.


Yu Zhang is a PhD candidate with the state key laboratory of robotics and system at Harbin Institute of Technology, China. She is broadly interested in modular self-reconfigurable robot development in the context of controlled systems.

Hongzhe Jin is an associate professor of Mechanical and Electronic Engineering at Harbin Institute of Technology. Hongzhe Jin received the B.A. degree in precision instrument engineering from the Harbin Institute of Technology in China. He received the M.A. degree and Ph.D. degree in electronic engineering from PUSAN National University. His research is in complex system dynamics, and their control theory and technology.

Jie Zhao is a Professor of Robots and control with State Key Laboratory of Robotics and System, Harbin Institute of Technology, China. He is broadly interested in design and control of industrial robots, medical robots and bio-robots.