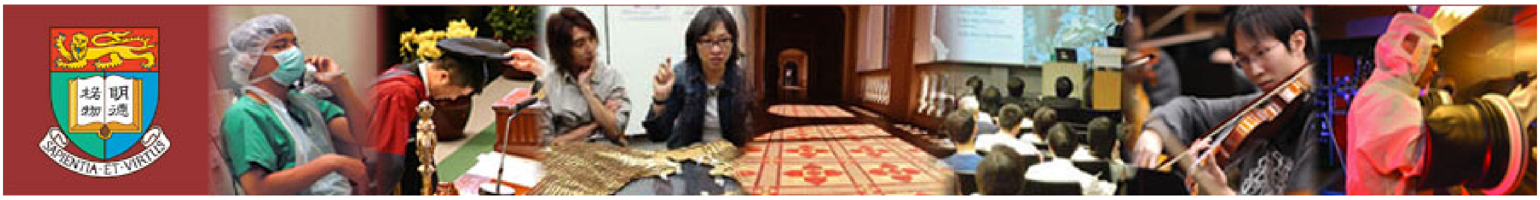


2013 10th IEEE International Conference on Control and Automation (ICCA)
Hangzhou, China, June 12-14, 2013

Distributed Algorithms for Shape Sculpting of Lattice-Arrayed Modular Robots via Hole Motion

Jingwei Zhu[1], Michael Z. Q. Chen[1] and Housheng Su[2]

***Abstract*— A self-reconfigurable modular robot can change its own shape by rearranging the connectivity of the modules of which it is composed. In this paper, we focus on a two-dimensional lattice-arrayed self-reconfigurable modular robotic system. Each module can move to a neighboring lattice under certain motion constraints, communicate with its neighbors and act upon local knowledge only. A scalable shape sculpting algorithm based on the manipulation of regularly shaped voids within the lattice ("holes") is given. We present detailed solutions to the conflict test and settlement problem encountered when applying this algorithm, and make improvement on the efficiency of shape sculpting. We believe that the algorithm can potentially generalize to 3D and scale to handle millions of modules.**

I. INTRODUCTION

Multi-agent systems have attracted much attention in recent years [1], [2]. Recently, interest in self-reconfigurable modular robots has been growing. A self-reconfigurable modular robot is believed to be versatile since it can change its own shape or structure by rearranging the connectivity of a large number of modules it is built from to adapt to new environments and tasks. Similar idea can be found in programmable matter [3], which refers to matter that has the ability to change its physical properties such as its shape in a programmable fashion. In such an application, the matter comprises fine-grained computing elements, fabricated by advanced semiconductor technology and nanoscale technology and usually used in extremely large quantity in one ensemble. Mass production of modules further cuts down the manufacturing costs of robots and makes quality control easier. Despite all the benefits, there are tough challenges in the mechanical design and distributed control algorithm design of such robotic systems.

Two different approaches have been developed to the mechanical design of a self-reconfigurable robotic system. In the first approach, modules are packed in some spatial crystal patterns to achieve desired shape. Modules in such a network can be compared to atoms in a crystal. They can move to neighboring lattices conforming to certain constraints, thus changing the connectivity of modules. As a result, the shape or structure of the robot is altered. Chirikjian [4] and Murata *et al.* [5] provided initial ideas and mechanical designs of such lattice-arrayed robotic systems. This kind of system shows great potential in self-assembly and self-repair, but is not suitable for realizing locomotion of the whole robot. The second approach was first proposed by Yim [6]. Modules are connected to form a chain and are able to reach any point in space. This chain architecture makes locomotion of the robot easier to be accomplished compared to the lattice architecture, while the self-reconfiguration step requires much more accuracy. Also there are hybrid architectures that combine and take advantages of the previous two architectures. Here, we focus on a lattice-arrayed modular robotic system.

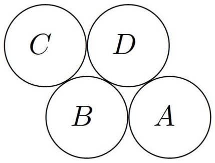


Fig. 1. Motion Constraints ("Catom"): *B*'s movement to position *D* requires a module in position *C* to pivot about and empty space in positions *A* and *D* to move through.

Motion planning to enable self-reconfiguration process is one of the most fundamental and challenging problems in the field of algorithm design for modular robotic systems. In a lattice-arrayed system, a module can move to one of its unfilled neighboring lattices under *motion constraints* (Figure 1) specified by the mechanical design. Motion constraints existing in almost all realizable module designs make the planning problem intractably difficult. And many previous algorithms tend to be trapped in local minima, for instance, [11], [15]. Methods to ease these problems can be broadly categorized into the metamodule-based method in [7], [8], [9] and the hole motion method in [10].

A metamodule is a group of modules that can be regarded as a unit during the planning process. It can easily absorb or recreate another metamodule without violating motion constraints. Through the absorb/recreate behavior at the perimeter, the contour of the robot is increased/reduced. And modules can be transferred from deletion regions to growth regions in the same way (Figure 2).

De Rosa *et al.* [10] use the randomized motion of regularly shaped voids (that is, holes) in a lattice as primitive operations for a two-dimensional shape formation. A hole is created by enclosing empty space at the perimeter in the growth region and moves through the mass like the molecules

This work was supported by the National Natural Science Foundation of China under Grants 61004093 and No. 61104140 and the Program for New Century Excellent Talents in University from Chinese Ministry of Education under Grant NCET-12-0215. Correspondence: MZQ Chen, mzqchen@gmail.com.

[1]Jingwei Zhu and Michael Z. Q. Chen are with the Department of Mechanical Engineering, The University of Hong Kong, Hong Kong.

[2]Housheng Su is with the Department of Control Science and Engineering, Huazhong University of Science and Technology, China.

978-1-4673-4708-2/13/$31.00 ©2013 IEEE

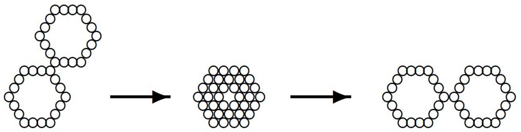


Fig. 2. Motion Plan in A Metamodule System: A metamodule has the ability to absorb/recreate other metamodules to reconfigure without violating motion constraints. This motion plan only relies on one neighboring metamodule.

of an ideal gas until it reaches the perimeter in the deletion region, where it creates "craters" and the contour is reduced. By treating a hole as an entity and solving the motion planning problem of holes, the motion constraints can be avoided. And because of the randomized motion of holes, the algorithm is not susceptible to entrapment in local minima. It is believed that this planner can be potentially generalized to 3D and be able to handle millions of modules. In this paper we mainly focus on distributed control algorithms based on the Hole Motion Planner.

II. PROBLEM FORMULATION AND PRELIMINARIES

Our work is based on the hardware design of "catoms" [16] (Figure 3). Modules are compactly arranged in the hexagonal array and each module has six neighboring lattices and up to six neighbors. The set of modules containing a module k, its neighbors, and their neighbors is defined as the *2-hop radius* of module k. A *hole* is defined as the logical entity formed by the absence of a module and its six neighbors. There are six possible directions for the motion of a hole. The 12 modules around a hole are defined as the *shepherd group*. These concepts are illustrated in Figure 4.

In the Hole Motion Planner, the two-dimensional coordinate space containing the modules is initially filled with equilateral triangles with a specific size. This size determines the resolution of contour that the robot can achieve. The initial and goal shapes are both properly decomposed into a group of triangles in the coordinate system. Triangles that do not change their state of occupation during the transformation from initial to goal shape are removed, while the rest become growth regions or deletion regions based on whether they overlap regions within the goal shape. Then the position information of these tri-regions in the shared coordinate system is transmitted to each module, provided that modules always have knowledge of their relative positions. After that no broadcast is needed.

If a module detects that it is on the perimeter of the mass (that is, it has at least one unfilled neighboring site and it does not belong to any shepherd groups) and is inside a tri-region (growth region/deletion region), it becomes a *growth/deletion node*. Before a growth node moves on to create a hole, it has to check whether the following conditions hold:

- There are no holes or shepherd groups within the node's 2-hop radius. Also there are no other growth nodes that are about to create holes which have conflict with the node (that is, the shepherd group of a hole should never overlap another shepherd group. If there is such an overlap, the overlapping shepherd groups can be regarded as "in conflict").
- There are exactly 11 other modules within the 2-hop radius of the growth node.

Fig. 3. Three magnetic 45mm planar catoms from [16].

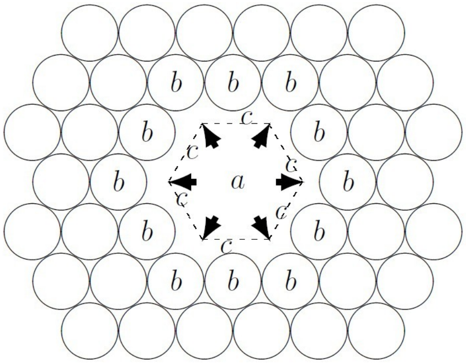


Fig. 4. Basic Layout: a) hole, b) shepherd group, c) six possible motion directions.

We develop a distributed control algorithm to test whether such a conflict exists and resolve the conflict if there is any.

If the above conditions are satisfied, the growth node starts to create a shepherd group with the help of neighbors within its 2-hop radius and a hole is generated on the perimeter. After creation, at each timestep a hole makes decision on direction of motion and shifts the three modules at the leading edge of the shepherd group (determined by the direction of motion) to the trailing edge, thus moving one step along the direction of motion (Figure 5). In De Rosa *et al*. [10]'s work, a hole's direction of motion in the ensemble is randomized. We introduce a "temperature field" inspired by the heat-based method in [11] to guide the motion of holes before they reach deletion nodes and create "craters" on the contour. The efficiency of shape sculpting can be greatly improved under the guidance of this field and when it is stabilized no *basin* (If the value of any module within a domain is smaller than that of any module on the boundary of that domain, the set of modules within that domain is defined as a basin.) inside the perimeter that might trap the holes would be created by the introduction of the field.

If a hole finally reaches deletion region and activates a deletion node, the deletion node destroys the hole and the contour of the robot is lowered, thus completing the life circle of a hole (Figure 6).

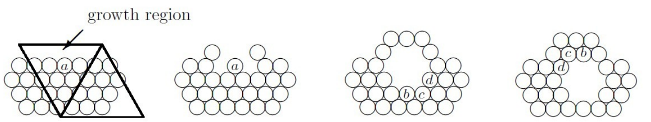


Fig. 5. Hole Creation and Hole Movement: The growth node a creates a hole on the perimeter. The three modules b, c, d at the leading edge (determined by the direction of motion) of the shepherd group are shifted to the trailing edge, thus shifting the hole one module in the direction of motion.

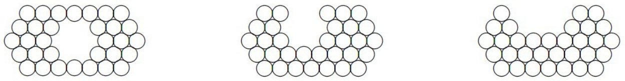


Fig. 6. Hole Deletion.

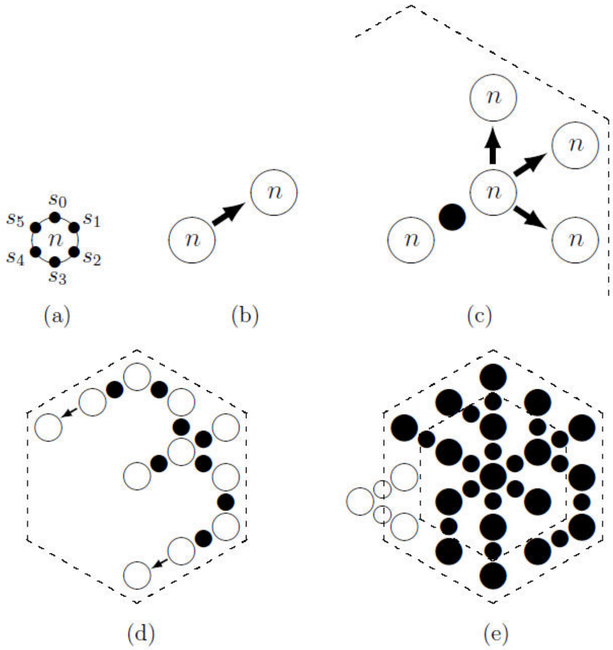


Fig. 7. Spanning Tree within The 2-hop Radius: a) A growth node selects a random identity n from a finite domain (each module has 6 interfaces for connection and communication). b) A growth node recognizes one of its neighbors as its branch and that neighbor gets the same identity. c) A branch in the second layer establishes new branches only in the third layer. d) Branches in the third layer establish new branches only in the third layer. e) Modules' attempts to establish branches can be frustrated by branches of other modules in the same spanning tree or in another spanning tree.

III. MAIN RESULTS

A. Algorithm for Conflict Test and Settlement

This algorithm is actually to deal with leader election problems among groups of modules. Difficulties of the design of this algorithm lie in that all the modules are identical, act upon the same local algorithm, but to finish the task of conflict test and settlement they have to play different roles and also cooperate with other modules in different roles.

The creation of a hole is started by a newly activated growth node. Modules (excluding the growth node) within the 2-hop radius of a growth node are categorized into *the second layer* and *the third layer*. The second layer modules are neighbors of the growth node. The third layer modules are the rest of the modules within the 2-hop radius. First, the growth node has to make sure that all the modules within its 2-hop radius can be recruited by itself to create a hole. Once recruited, modules can no longer be recruited by other growth nodes until they are released from the shepherd group. The growth node sets up connections with its neighbors by recognizing them as its *branches*. If a module m_x is recognized as a branch by another module m_y, m_x would also recognize m_y as its *source*. Likewise, these branches set up connections with their neighbors within the 2-hop radius of the growth node according to the rules shown in Figs. 7(c) and 7(d), and finally a spanning tree within the 2-hop radius is formed. The root of each spanning tree is always a growth node. If every module within the 2-hop radius of a growth node is a node of the same spanning tree, these modules would be recruited by the growth node and move on to create a hole without any conflict. Otherwise, negotiation to resolve the conflict has to be started. After negotiation, some of the growth nodes and their spanning trees would be sacrificed (that is, deactivated) while others survive and create holes when there is no conflict.

During the process of building spanning tree within the 2-hop radius, it is very possible that when a module m_x attempts to recognize a neighbor m_y as its branch, m_x finds out that m_y has already been a branch of another module (Fig. 7(e)). There are two possibilities for such a situation:

- m_y shares the same root as m_x (that is, they are in the same spanning tree).
- The root of the spanning tree m_y is different from the root of m_x.

However, in most cases m_x as a module cannot tell the difference between the two possibilities. We propose a solution where the growth node stores position information of all the modules within its 2-hop radius while other modules do not have to save that kind of information. For each module except the growth node, if its attempt to establish a branch fails, it records that event in a "report" and send the report back to its source later. That report will be passed on and finally reach the growth node since the root of spanning tree is always a growth node. Then, based on these conflict reports and the position information, the growth node may conclude whether the conflicts are internal or external. If the conflict is internal (that is, a module attempts to recognize another module which is in the same spanning tree as its branch), it will simply be neglected. Otherwise negotiation should be started to resolve the conflicts.

The so-called negotiation is basically a leader election among groups. One of the candidates survives and is elected as the leader while others are sacrificed. Since there exists no terminating algorithm for electing a leader in an asynchronous anonymous network, a *Las Vegas* algorithm is a good option.

The size of the state space of a module in the Hole Motion

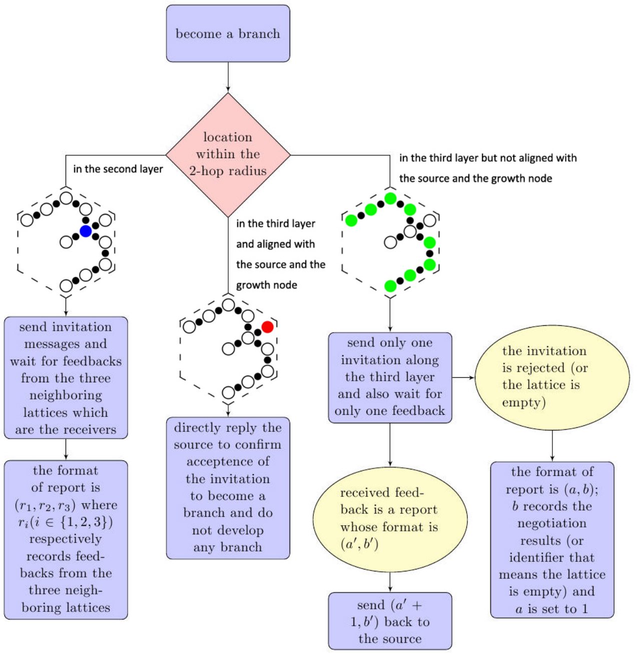

become a branch
location within the 2-hop radius
in the second layer
in the third layer and aligned with the source and the growth node
in the third layer but not aligned with the source and the growth node
send invitation messages and wait for feedbacks from the three neighboring lattices which are the receivers
the format of report is (r_1, r_2, r_3) where $r_i (i \in \{1, 2, 3\})$ respectively records feedbacks from the three neighboring lattices
directly reply the source to confirm acceptence of the invitation to become a branch and do not develop any branch
send only one invitation along the third layer and also wait for only one feedback
the invitation is rejected (or the lattice is empty)
received feedback is a report whose format is (a', b')
send $(a' + 1, b')$ back to the source
the format of report is (a, b); b records the negotiation results (or identifier that means the lattice is empty) and a is set to 1

Fig. 8. Report formats in different locations within the 2-hop radius. (Note that a in (a, b) is actually the number of modules in a continuous branch "chain" in the third layer.)

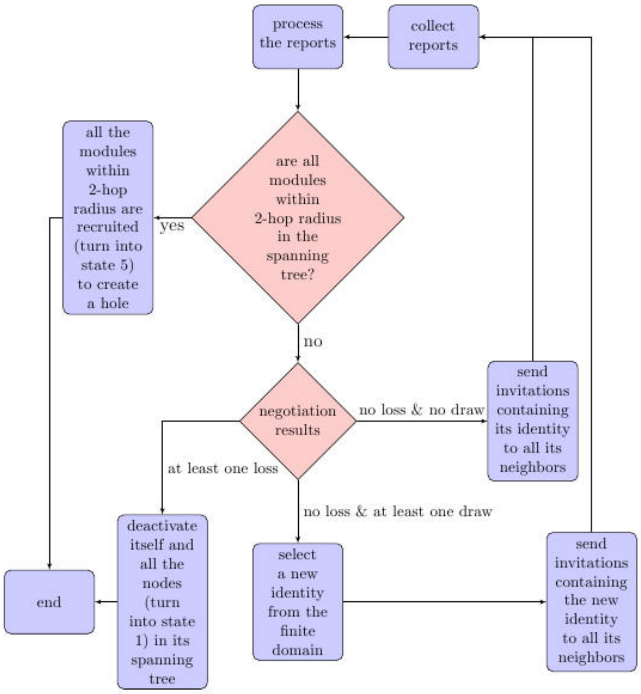

process the reports
collect reports
all the modules within 2-hop radius are recruited (turn into state 5) to create a hole
yes
are all modules within 2-hop radius in the spanning tree?
no
negotiation results
no loss & no draw
send invitations containing its identity to all its neighbors
at least one loss
no loss & at least one draw
deactivate itself and all the nodes (turn into state 1) in its spanning tree
end
select a new identity from the finite domain
send invitations containing the new identity to all its neighbors

Fig. 9. Steps taken by the growth node after processing the reports.

Planner is 7:

- State 1: Passive state. A module in State 1 serves as a conductor to propagate "heat" to their neighbors. They are the key components to form the "temperature field", which will be discussed in detail in next section. Initially all the modules are in State 1.
- State 2: Growth node.
- State 3: The second layer. A module in the second layer is one of the neighbors of a growth node.
- State 4: The third layer. A module in the third layer is in the neighboring lattice (excluding the second layer and the growth node) of the second layer.
- State 5: Transient state. A module in transient state is truly recruited by a growth node and is ready for operations of hole creation (that is, the conflict test and settlement steps are finished).
- State 6: Shepherd group.
- State 7: Deletion node.

After being activated, each growth node selects a random identity from a finite domain. If the growth nodes try to set up connections with their neighbors, invitation messages containing the selected identity and its own state are sent. Initially, each growth node sends invitation messages through all the interfaces that are connected with other modules. Modules in State 1 or State 7 are free to become branches. Once recognized as a branch, a module sets its own identity to the one contained in the invitation message, changes its state according to the state of the sender, and it also sends out invitation messages containing its own identity to set up new connections with others (rules of establishing new branches are illustrated in Figure 7). Thus, modules belonging to the same spanning tree share the same identity. Note that the invitation message sent by branch also contains information about the state of sender so that the receiver could learn which layer it should be in and they change their state accordingly. Additional information is needed in the invitations sent by modules in State 3 to modules in the third layer so that, if recognized as branches, they can learn whether they are aligned with the growth node and their source. This determines the format of reports these third layer modules produce, which will be discussed in detail later.

If a module m_x sends an invitation to a neighbor m_y which is already a branch of another module, m_y sends back a message containing its own identity after receiving that invitation. m_x calculates the results of negotiation based on the identity contained in the reply. If the received identity is larger than its own, m_x loses the negotiation; If the received identity is equal to its own, m_x takes a draw; If the received identity is smaller, m_x wins. Note that if the invitation is sent to a module in State 5 or State 6, it will be rejected and the negotiation result for the sender of invitation is always a loss. The negotiation results are saved in the memory of m_x. After sending invitations, each new branch waits for feedbacks and prepares for a report which will be sent back to its source. That report might contain results of negotiation or reports from branches. Branches in different locations within the 2-hop radius would produce reports in different formats. There are three cases in the report format as shown in Figure 8.

When finishing collecting the feedbacks from all the invitation receivers, each branch sends back the report to their source. Finally, a growth node gets the necessary information about every node of its spanning tree. Based on the received reports and map of its 2-hop radius, a growth node can know whether a module within its 2-hop radius is in its spanning tree or not. As mentioned above, some of the invitations would be rejected by modules in the same spanning tree, and these negotiation results (should always be a draw) are neglected by the growth node. And if there exist some modules within the 2-hop radius belonging to other spanning trees, the negotiation results are learnt from the reports and recorded by the growth node. After processing the reports, a growth node behaves according to one of the steps shown in Figure 9.

For a module which has already been a branch, if it receives a new invitation from its source, it would update its identity, send out new invitation containing the new identity and collect feedbacks for the report. When the report is completed, it is sent back to the source.

Such a loop will continue in each spanning tree until all the conflicts are finally settled.

Remark. The algorithm is implemented in a Python-based simulator and it works well in most cases. However there is one exception. In some situations, some of the modules do not have a path to the growth node within the 2-hop radius. Our algorithm cannot work here since conflict test and settlement all rely on the spanning tree within the 2-hop radius. Future work could be improving our algorithm to solve such a problem or finding methods to prevent such a situation from occurring.

B. Algorithm For Forming Temperature Field

In this section, we present a distributed algorithm for forming "temperature field" among the modules, which is inspired by the heat-based method in [11]. The whole robot is a conductor and each module can be approximated as a differential element which yields to the heat equation. Modules in the shepherd group are heat sources whose temperature is constant. Deletion nodes are heat sinks and its temperature is also constant but lower than that of heat sources. Heat exchange takes place between two neighboring modules at each timestep and the exchanged quantity of heat is proportional to the temperature difference. Let $T_i(t)$ be the temperature of a module i whose $state = 1$ at time t, N_i be the set of module i's neighbors whose $state \in \{1, 6, 7\}$, and c_{ij} be the thermal conductivity between two modules ($c_{ij} > 0$):

$$T_i(t+1) = T_i(t) + \epsilon \sum_{j \in N_i} c_{ij}(T_j(t) - T_i(t)), \qquad (1)$$

where $0 < \epsilon < 1/\Delta$ and Δ is the maximum degree of the network.

After the initial turbulence, the temperature field is stabilized in several timesteps and can be used to guide holes. The states (temperature value) of all the modules converge to a unique and constant vector. Each time a hole is created or makes a move, new turbulence is introduced into the field and it takes time to stabilize. Therefore, the decision making and travel frequency of a hole should be much lower than the temperature updating frequency of passive state modules.

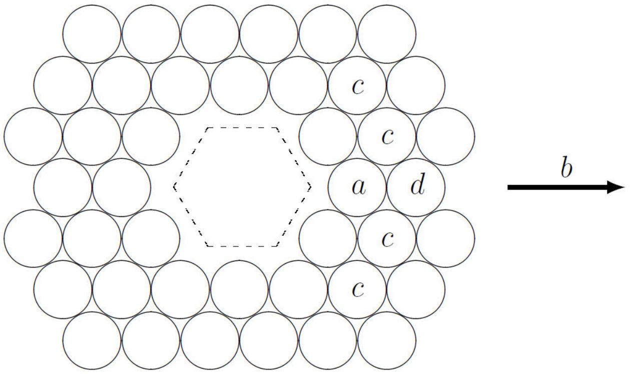
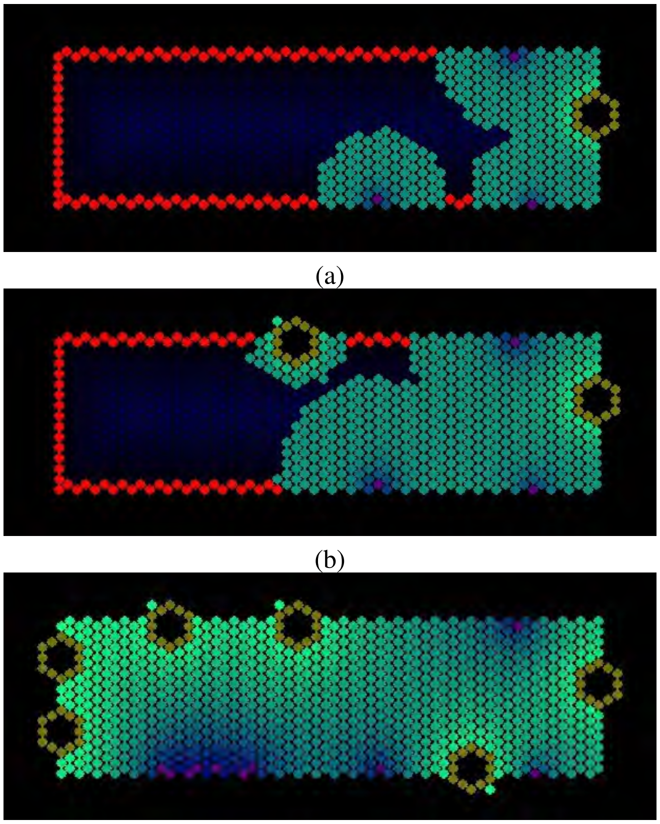


Fig. 10. Relevant Modules for Navigation through Temperature Field: a) module at the vertices of the hexagonal shepherd group, b) the direction of motion module *a* represents and if module *a* is elected as the leading module the hole would move one step in this direction, c) and d) five lattices in the direction of motion to be checked by module *a*, d) If there is no deletion node in the five lattices but these lattices are all filled by modules in State 1, the temperature value of *d* is saved by *a* for future comparison.

With this algorithm, basins of the temperature field can only be found on the perimeter when the field is stabilized. Since no deletion nodes (heat sink) exist within the perimeter, if there are basins inside the perimeter when the temperature field is stabilized, these basins cannot contain any deletion nodes. As the thermal flux into the basins is positive, temperature of modules in the basins can not get stabilized without a deletion node. On the other hand, those basins on the perimeter are exactly what we use to lead holes to deletion nodes.

If one of the six modules at the vertices of the hexagonal shepherd group is chosen to be the leading module, it means that the hole would move towards that module (that is, the three modules including the leading module at the leading edge of the shepherd group would be shifted to the trailing edge). For the six modules at the vertices, if there is at least one deletion node in the five corresponding lattices in the direction of motion (Figs. 10(c) and 10(d)), they save the temperature value of the deletion node for later comparison and decision making and become a candidate for leading module; otherwise, if there is no deletion node in the five lattices but these lattices are all filled by modules in State 1, they save the temperature value of one of their neighbors which is in the direction of motion they represent (Fig. 10(d)) and also become a candidate; if neither of the two conditions is satisfied, they cannot be chosen as the leading module. When all the six modules get ready (that is, finish the checking steps mentioned above), the module which gets the lowest temperature value would be chosen as the leading module. (It is possible that some of the modules would get exactly the same lowest value, and a random leading module would be chosen from them.) A probabilistic leader election algorithm for breaking symmetry in anonymous rings developed by [13] can be applied here perfectly to find the module with lowest value and break



(a)

(b)

(c)

Fig. 11. Simulation of Temperature Field Formation: a) and b) Initially all the modules are in State 1 and their temperature value is set to 0. Perimeter of the ensemble is marked with red color. After holes or deletion nodes are created on the perimeter, turbulence propagates through the robot body. c) The temperature field can be stabilized in several timesteps. There are no basins of the field existing inside the perimeter when the field gets stabilized. Deletion nodes are global minima on the perimeter.

symmetry if necessary.

With such a strategy, when two holes get close, the region between them would be heated, which prevents the two holes from moving closer. Similarly, in most cases a hole rebounds back before reaching the perimeter. Since deletion nodes (heat sinks) are always global minima in terms of temperature value, holes are attracted to deletion nodes. Obviously path planning with temperature field is much more efficient than merely depending on random motion of holes. Although all the communication and calculation are local and distributed, a hole can find an optimal path leading to the deletion nodes and take evasive action just like it has global information of the ensemble.

We implemented the algorithm in a Python-based simulator. We set the thermal conductivity to 0.1, set ϵ to 0.5, set the temperature of shepherd group to 99999 and the temperature of deletion node to 0. The relation between the 8-bit RGB color of modules and their temperature T is $RGB\ color = [0, \frac{255T}{99999}, 130]$. Shepherd groups are denoted by brown color and deletion nodes are denoted by purple. The process of forming the temperature field is shown in Figure 11.

IV. Conclusion

We have presented the design of some important distributed algorithms for shape-sculpting of lattice-arrayed modular robots via hole motion. We also introduced a temperature field to guide the holes so that the efficiency of shape-sculpting is improved compared to the randomized motion of holes. However, there still exist limitations in our algorithms and further simplification could be made. Our work is an initial attempt to creating a fully distributed and scalable algorithm for self-reconfigurable modular robotic system and we hope it could provide inspiration for later work in the algorithm design for modular robots or other distributed control systems.

References

[1] H. Su, X. Wang and Z. Lin, "Flocking of multi-agents with a virtual leader," *IEEE Transactions on Automatic Control*, vol. 54(2), pp. 293–307, 2009.

[2] H. Su, Z. Rong, M.Z.Q. Chen, X. Wang, G. Chen, and H. Wang. "Decentralized adaptive pinning control for cluster synchronization of complex dynamical networks," *IEEE Transactions on Cybernetics*, vol. 43, no. 1, pp. 394–399, 2013.

[3] S. Goldstein, J. Campbell, and T. Mowry, "Programmable matter," *IEEE Computer*, vol. 38, pp. 99–101, 2005.

[4] G. Chirikjian, "Kinematics of a metamorphic robotic system," *IEEE Intl. Conf. on Robotics and Automation*, pp. 449–455, 1994.

[5] S. Murata, H. Kurokawa, and S. Kokaji, "Self-Assembling machine," *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pp. 441–448, 1994.

[6] M. Yim, "A reconfigurable modular robot with many modes of locomotion," *Proc. of JSME Intl. Conf. on Advanced Mechatronics*, pp. 283–288, 1993.

[7] C. Unsal and P. Khosla, "A multi-layered planner for self-reconfiguration of a uniform group of i-cube modules," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[8] D. Christensen, E. Ostergaard, and H. H. Lund, "Metamodule control for the atron self-reconfigurable robotic system," *Proc. of the 8th Conference on Intelligent Autonomous Systems (IAS-8)*, 2004.

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

[10] M. De Rosa, S. Goldstein, P. Lee, J. Campbell, and P. Pillai, "Scalable shape sculpting via hole motion: motion planning in lattice-constrained modular robots," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1462–1468, 2006.

[11] M. Yim, Y. Zhang, J. Lamping, and E. Mao, "Distributed control for 3D metamorphosis," *Autonomous Robots*, vol. 10, no. 1, pp. 41–56, 2001.

[12] M. P. Weller, M. D. Gross, and S. C. Goldstein, "Hyperform specification: designing and interacting with self-reconfiguring materials." *Personal and Ubiquitous Computing*, 2011.

[13] A. Itai and M. Rodeh, "Symmetry breaking in distributed networks," *Information and Computation*, 88(1): 60–87, 1990.

[14] W. Fokkink and J. Pang, "Simplifying Itai-Rodeh leader election for anonymous rings," *Proc. of the Fourth International Workshop on Automated Verification of Critical Systems*, Electronic Notes in Theo. Comp. Sci., vol. 128 , pp. 53–68. Elsevier, 2005.

[15] H. Bojinov, A. Casal, and T. Hogg, "Emergent structures in modular self-reconfigurable robots," *Proc. of the IEEE International Conference on Robotics and Automation*, 2000.

[16] B. Kirby, J. Campbell, B. Aksak, P. Pillai, J. F. Hoburg, T. C. Mowry, and S. Goldstein, "Catoms: Moving robots without moving parts," *AAAI (Robot Exhibition)*, pp. 1730–1731, Pittsburgh, PA, 2005.

[17] R. Olfati-Saber and R. M. Murray, "Consensus problems in networks of agents with switching topology and time-delays," *IEEE Trans. Autom. Control*, vol. 49, no. 9, pp. 1520–1533, Sep. 2004.