Proceedings of the 2002 IEEE
International Conference on Robotics & Automation
Washington, DC • May 2002

A Complete, Local and Parallel Reconfiguration Algorithm for Cube Style Modular Robots

Serguei Vassilvitskii[1]
Cornell University
sv39@cornell.edu

Mark Yim
Palo Alto Research Center
yim@parc.com

John Suh
Palo Alto Research Center
jwsuh@parc.com

Abstract

We present a complete, local, and parallel reconfiguration algorithm for metamorphic robots made up of *Telecubes*, six degree of freedom cube shaped modules currently being developed at PARC. We show that by using 2x2x2 meta-modules we can achieve completeness of reconfiguration space using only local rules. Furthermore, this reconfiguration can be done in place and massively in parallel with many simultaneous module movements. Finally we present a loose quadratic upper bound on the total number of module movements required by the algorithm.

1 Introduction

Modular Self Reconfigurable Systems consist of many identical robots that are very limited in their actions. As the number of modules in a system increases, the range of behaviors of the group of robots grows exponentially. The task of self-reconfiguration is important for developing self-sufficient systems. The overall system can reconfigure itself to help accomplish certain tasks such as locomotion, object manipulation and sorting, or interaction with other systems, especially when there is a need to adapt to the environment.

Previous research has established that by grouping single modules into groups or meta-modules each unit in the system increases its number of degrees of freedom and the reconfiguration tasks are simplified [4, 5, 7, 10]. However using meta-modules limits the granularity of the possible configurations. Rus and Vona require 4x4 meta-modules for complete 2D reconfiguration for expanding cube style modules, while Nguyen et al. explore the possibility of 36 membered meta-modules for 2D reconfiguration with hexagonal modules. We propose meta-modules composed of 8 modules for 3D reconfiguration and guarantee completeness in the parallel reconfiguration.

We proceed by motivating the need for a new reconfiguration algorithm. In section 4 we describe the hardware platform currently being developed at PARC (formerly Xerox PARC) that inspired our work. We then describe the new locomotion primitives for the 2x2x2 meta-modules. In section 7 we present the self-reconfiguration algorithm along with its analysis and correctness results.

2 Related Work

The problem of reconfiguration for modular self-reconfigurable robotic systems has received increased interest. This work includes [[1]-[8], [12]-[15]]. In [14] Walter et al. focus on limiting communication between the individual modules. Pamecha and Chirikjian explore probabilistic techniques such as Simulated Annealing in [6]. Rus and Vona have proposed the use of meta-modules to guarantee completeness of reconfiguration spaces (the space of all possible configurations) in [7, 8]. Their melt grow algorithm uses 4x4 meta-modules to solve the general reconfiguration problem in two dimensions.

More recently as the focus has shifted on decentralized control and parallel actuation, Butler et al. introduced Cellular Automata for distributed control[2], along with the PacMan algorithm for concurrent actuation by several modules [1].

3 Motivation

The ideal reconfiguration algorithm would be complete for all possible shapes, allow for more than one set of concurrent module movements and be completely autonomous. Each of the above algorithms lacks one of the above properties: The PacMan algorithm[1] along with the work by Walter et. al [14], is not complete, while Melt-Grow [8] is not distributed and does not allow for parallel actuation.

To incorporate all of the desirable features into one algorithm, we begin by using 2x2x2 meta-modules. Vassilviskii et. al [12] prove completeness for reconfiguration using the 8 membered meta-modules and the algorithm presented here follows this work. We simplify the planning portion of the algorithm while retaining completeness for the meta-module configuration space. We provide an algorithm which performs in place parallel distributed reconfiguration in worst case quadratic time.

0-7803-7272-7/02/$17.00 © 2002 IEEE

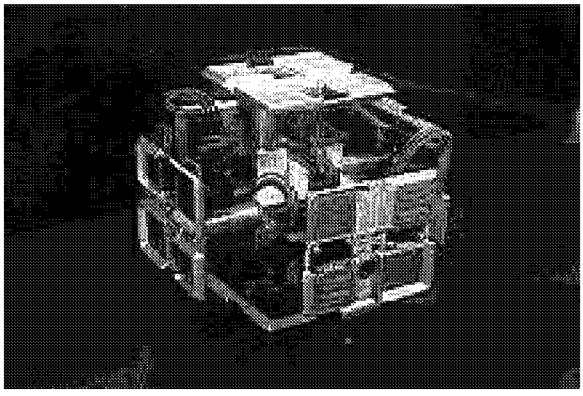

Figure 1: One telecube module.

4 Basic Operations of the Telecube Module

We look at the Telecube generation of modular robots designed prototyped and currently being constructed at PARC [11]. Similar in design to the Crystalline modules used by Rus [7, 9], these modules are cube shaped. The Telecube modules have the ability to independently extend out each of the 6 faces of the cube where as the Crystalline modules extend out of only 4. These extensions and retractions provide the modules' only form of motion. A picture of a module is shown in Figure 1.

The module arms can extend independently up to half of the body length, giving the robot an overall 2:1 expansion ratio along each dimension. A latching mechanism on the plates on the end of each arm enables two aligned modules to connect to each other. For power routing, communication and alignment reasons, the modules must remain globally connected in one connected component at all times. While the modules are in construction, we have built a simulator to develop and test reconfiguration algorithms.

The modules have the following low-level primitives:

- *ExtendArm(Direction)*: If there is room to extend the arm in *Direction*, extend the arm.
- *RetractArm(Direction)*: Retract the arm in *Direction*, attempting to first disconnect from neighbor if connected.
- *Connect(Direction)*: If there is a neighboring module in *Direction*, latch to that module.
- *Disconnect(Direction)*: If there is a module is currently latched in *Direction*, break the connection with the neighbor.

From these primitives, we can build more complicated actions, such as *Move(Direction)*. The explicit sequence of actions that allows a module to move along a given direction is illustrated in Figure 2. A module can pull towards a neighbor by retracting its arm, push away from a neighbor by expanding its arm, or simultaneously retract its front arm and expand its back arm, effectively "sliding along its arms" in a given direction. Prior to moving, the module:

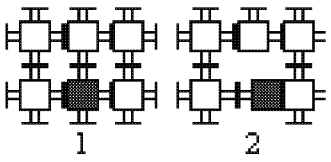

Figure 2: The shaded module moves one arm length

1. confirms that it has at least one neighbor along the direction of motion on which it can push or pull,
2. ascertains that it is moving into free space,
3. disconnects from all neighbors perpendicular to the direction of movement

At any point during this process, the movement can fail, in which case the module reverses what has been done so far and returns to its original state.

Each module is also given simple sensing and communication abilities. Modules can send messages through their faceplates to their immediate neighbors using a low bandwidth IR link. Each module can also gauge the extension of each faceplate, read the contact sensor on each of the faces, and determine whether it is latched to a neighboring module.

5 Simulator

To proceed with algorithm development, we have built a simulator for the Telecube system. The simulator written in Java limits each module to the same exact primitives as those of the true physical module. Each module in the simulator has the opportunity to move once per time step. To simulate the asynchronous qualities of the system, the order in which the modules move is randomized and is different each turn.

For simplicity reasons we limit the state of the module arms to either fully extended or fully contracted. We also assume the module arms to be infinitely rigid so that they occupy the nodes of a perfect lattice structure at all times. During each simulated move actuator torque and stiffness constraints are imposed to make sure the modules are not dragging more than two other modules per move. The global connectivity constraint is also checked before every disconnect request.

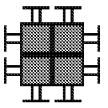

Figure 3: A schematic of one meta-module. Only one layer shown.

The simulator has a Java 3D User Interface allowing us to capture individual frames and animations of full reconfigurations.

6 Primitives

6.1 Meta-Modules

To achieve completeness of reconfiguration, we use meta-modules composed of 8 individual Telecubes. The cubes are arranged in a tight cube with their arms fully retracted. Each cube belongs to one and only one meta-module during reconfiguration, though two meta-modules may exchange individual cubes during the process. A schematic of one layer of a meta-module is shown in Figure 3.

We define three locomotion primitives: *Move, Roll* and *S-Roll.* We envision the meta-modules located in a three dimensional lattice, with each meta-module having the coordinates (x, y, z). An adjacent meta-module has one of the coordinates differing by 1, for example $(x, y, z + 1)$.

6.2 Move

The primitive Move(dirMove) is a natural extension of the Move primitive for the individual modules, it moves the meta-module one step in the given direction. For example, Move(EAST) would result with a meta-module at (x, y, z) to move to position $(x + 1, y, z)$. The moving meta-module requires two meta-modules to move upon. The exact sequence of the moves by the individual modules is shown in Figure 4.

6.3 Roll

The Roll(dirRoll, dirSubstrate) has no analog on the lower level. The Roll allows for one meta-module to "roll" around a corner of another meta-module. For example, Roll(EAST, SOUTH) results in a Meta-Module at (x, y, z) to move to position$(x + 1, y - 1, z)$. The Roll primitive requires for only one meta-module in the neighboring space indicated by the dirSubstrate direction. The exact sequence of moves by the individual modules during an execution of a Roll primitive is shown in Figure 5.

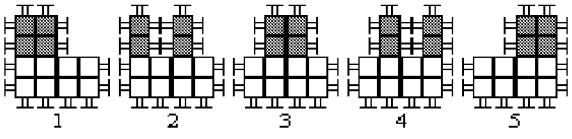

Figure 4: A meta-module executing Move(EAST). Only one layer shown.

6.4 Tunnelling

While looking at the exact sequence of individual motions that result in a meta-module move or a roll, it is important to notice that each has a clear midpoint when half of the active meta-module is occupying the previous position and exactly half is occupying the new position. These correspond to state 3 for *Move* and state 7 for *Roll.* Also note that two adjacent meta-modules can remain connected to each other even if there is a gap of one module in width in between them by extended the arms between them. Using these two facts, we can string a number of Move and Roll operations together on adjacent meta-modules so that they would be able to move without globally disconnecting from each other.

For example, assume modules a, b, c are arranged in a horizontal line.(Figure 6(i)) We assign the following moves to them:

a: *Move(EAST)*;

b: *Roll(NORTH, EAST)*;

c: None;

The execution would be as follows: Module b begins and completes the first half of the Roll(ii). At this point, it stops and waits for module a to complete the first half of the move. Once a completes the first half of the move(iii), b finishes its roll(iv), after which a finishes its move(v). There are two ways to look at the result, we can either say that the hole which b now occupies propagated through the structure to a or that a has tunnelled through the structure. We can extend this to longer chains of meta-modules by sequencing *Move(EAST)* moves.

We can see that the structure remains globally connected at all times, and will remain globally connected for arbitrarily long such sequences. There exists, however, one special case, Consider the example shown in Figure 7(i) and (ii). It appears that we are stuck at (ii), and we need a new low level locomotion primitive.

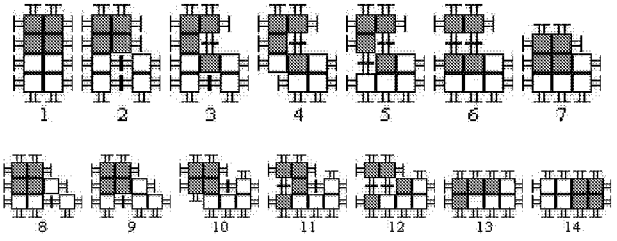

Figure 5: The shaded meta-module executing Move(EAST, SOUTH). Only one layer shown. The other layer moves identically.

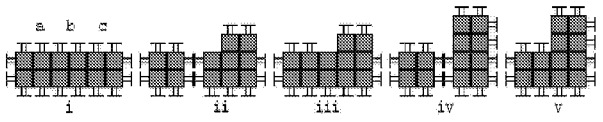

Figure 6: An example of tunnelling by meta-module A through meta-modules B and C

6.5 S-Roll

An S-Roll is the rarest primitive used, it requires an exact sequence of operations before it such as the one seen above. However, the solution is fairly trivial. Since we know that there is a meta-module following this path (else an S-Roll would not occur), we can switch two modules with the meta-module behind us. This is demonstrated in Figure 7 (iii) and (iv). At the time the configuration reaches state (iv), the next meta-module can continue with its motion.

7 Reconfiguration

As in [5] it is assumed the initial and final configurations overlap by at least one meta-module. On the whole, the algorithm is going to perform as follows:

- Select a module that can begin motion
- Plan a route for that module through the structure
- Execute the preplanned motions.

7.1 Selection

Each meta-module maintains aa distance value, δ, which keeps track of the minimum manhattan distance through the structure from this module to a module that is already in place, i.e. to a module that is in the final structure. (We note that the final structure is always connected during the reconfiguration). This value is easily maintained throughout the course of reconfiguration by the following:

δ = `Min(`δ `of neighbors)+1;`

Theorem 1. Given a meta-module, M, in a connected structure, S, If $\delta_M \geq \delta_N$ where N is a neighbor meta-module of M then $S - M$ is a connected structure.

Proof. If $\delta_M \geq \delta_N$ for all neighbors N then there exists a path for all N to the final structure that does not go through M. Since the final structure is connected within itself, the structure maintains a single connected component with M removed.

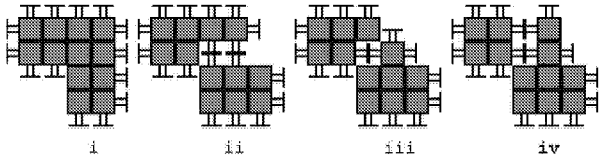

Figure 7: (i) The corner meta-module rolls to the East, the leftmost meta-module moves East. (ii) The tunnelling appears to be stuck. (iii)-(iv) After an S-Roll the reconfiguration continues.

Thus a meta-module is free to move once its distance value is at least as great as that of its neighbors.

7.2 Planning

Once a meta-module knows that it is free to move, it must plan a path through the structure to fill in one more node of the final structure. This path will consist of *Move*, *Roll*, and *S-Roll* commands and is guaranteed to exist since the initial and final configurations overlap and the system has one connected component.

To plan a move, we follow the technique similar to the PacMan algorithm [1]. We make one improvement, that of using exponential iterative deepening search instead of pure depth first search to find a path to the goal. In an iterative deepening approach, one searches first all of the nodes with depth less than 1, then all of the nodes with depth less than 2, then less than 4 etc. Thus, although several of the nodes are searched twice, we are much more likely to find a solution with a lowest depth, resulting in a quicker reconfiguration.

Once the path through the structure is generated, it can be trivially converted into a sequence of *Move*, *Roll*, and *S-Roll* directions which are assigned to the modules along the way.

7.3 Execution

The algorithm divides the meta-modules into two groups during its execution. The first is the active meta-modules identified in the selection step. They initiate the planning sequence as above. The second group is the passive meta-modules which act as part of the structure, but are not actively planning their own path. Rather, they are following the orders given to them by the active modules during their path planning as to where to move. These meta-modules may have several directions in which they have been told to move (or roll). It remains an open question whether there exists a heuristic to rearrange the order of these motions so as to minimize the total reconfiguration time of the overall structure.

8 Analysis and Discussion

We argue that the algorithm presented above is parallel, local and complete. The fact that many modules may be undergoing actuation at the same time is the simplest to see. If there are two modules on the opposite ends of the structure that have their δ values higher than their neighbors', nothing prevents them both from beginning to plan their path through the structure. Likewise, if there are two meta-modules which can begin their Roll or Move motions, indepen-

dently from each other nothing will limit them in their goal. Thus this algorithm is highly parallel.

All of the rules made by the modules are local rules. A module checks whether or not the planning stage can begin by consulting only its neighbors, a module propagates the planning request only to its immediate neighbors, and a module checks if it is safe to begin or continue actuation only by contacting its local neighbors. Thus there is never a centralized control point necessary for reconfiguration. This point of the algorithm is of key use in upgrading this system to a fault tolerant system since disabling any one part of the overall structure will not result in the complete loss of functionality for the remaining active modules.

Finally, we argue that the any shape composed of fully compacted meta-modules can reconfigure into any other such shape using the above algorithm.

Lemma 1. While the reconfiguration is not complete, there exists one meta-module, M whose δ value is at least as large as that of its neighbors.

Proof. Suppose such a meta-module does not exist. This means that for any meta-module, there exists one neighbor whose δ is strictly larger, which must in turn have a neighbor whose δ is strictly larger than its δ. This must continue implying δ is unbounded. However, δ is clearly bounded by the number of meta-modules, N. Therefore, such a meta-module M exists.

Lemma 2. Let M be the meta-module such that δ_M is higher than δ_N where N is a neighbor of M. Then there must exist a place adjacent to another meta-module M' which is currently unoccupied but sits in the final structure. Furthermore, there exists a path through the structure between M and M'.

Proof. (1). Since the number of meta-modules in the initial and the final structure is identical (no meta-module can be created or disassembled), if there exists a meta-module that is not in place, there must exist a place in the final structure which is not currently filled. (2). Since the both the initial and final structure must remain globally connected at all times, the union of the initial and final structures has one connected component, so there must exist an M' adjacent to an unfilled space from (1), and there always exists a path between M and any other meta-module M'.

Lemma 3. Given any two modules M and M', an empty space, H, adjacent to M', and a path between M and H through the structure, the module at M can tunnel through the structure and emerge at H.

Proof. Any path through the structure can be decomposed into a sequence of *Roll*, *Move* and *S-Roll* operations to be performed by meta-modules. Since the global connectivity constraint is never violated during tunnelling, the meta-modules on the path M,...,M' can execute their movements, which would result in the meta-module M emerging at H.

Theorem 2. Any connected structure of N meta-modules can reconfigure into any other connected structure of N meta-modules in place, in quadratic time, as long as the two structures have at least one meta-module in common at the start.

Proof. While there are meta-modules which are not in place, we can invoke Lemma 1 to find them, Lemma 2 to find the path to an empty space and Lemma 3 to tunnel down the path to fill this space. Since there are N meta-modules total, and the structures share at least 1 meta-module in common, the above process can be repeated at most $N-1$ times. Each tunnelling move may involve at most N meta-modules and each primitive takes O(1) time. Therefore, the total runtime for the algorithm is $O(N^2)$.

Furthermore, since all of the movements of the meta-modules are restricted to be contained within the initial or the final structure, the algorithm performs this reconfiguration in place.

We note that the time analysis above does not take into consideration two meta-modules moving simultaneously, and thus we believe that the quadratic time bound presented above is not a tight bound. We believe that in practice the worst case upper bound may be significantly lower, however, the analysis to demonstrate a tight upper bound remains an open problem.

9 Results

In Figure 8, we present snapshots of the algorithm on a reconfiguration from a flat sheet of meta-modules to a table. Each colored cube is a meta-module consisting of 8 individual Telecubes. There are 20 meta-modules, resulting in 160 Telecubes. The algorithm required a total of 311 individual time steps to complete the reconfiguration. The 311 is significantly smaller than the worst case quadratic bound on the meta-modules of 400.

10 Conclusion

The reconfiguration algorithms for metamorphic systems have lacked at least one of the desired properties: local decision making, completeness of reconfiguration or parallel execution. We have presented an algorithm which possesses all of the three qualities and is guaranteed to execute in place in worst case quadratic time. We resort to the use of meta-modules, as otherwise the space of possible configuration can be partitioned into classes with no configuration possible between members of different classes.

Acknowledgements: This work is supported in part by DARPA contract MDA972-98-C-0009.

References

[1] Butler, Z., Byrned, S., Rus, D.: Distributed motion planning for modular robots with unit-compressible modules, Proceedings of the 2001 Conference on the International Robotics Systems.

[2] Butler, Z., Kotay, K., Rus, D., Tomita, K.: Cellular Automata for Decentralized Control of Self-Reconfigurable Robots, in Proceedings of the 2001 IEEE Int. Conference of Robotics and Automation workshop on Robotic Modular Robots.

[3] Kotay, K. Rus, D.: Motion Synthesis for the Self-reconfiguring Molecule. Proceedings of the 1998 International Conference on Intelligent Robots and Systems.

[4] Nguyen, A., Guibas, L., Yim, M.: Controlled Module Density Helps Reconfiguration Planning, New Directions in Algorithmic and Computational Robotics. A. K. Peters 2001. 23 - 36.

[5] Pamecha, A. and Chirikjian, G.: A Useful Metric for Modular Robot Motion Planning, JHU Technical Report, RMS-9-95-1.

[6] Pamecha, A. and Chirikjian, G.: A Bounds for Self-Reconfiguration of Metamorphic Robots, JHU Technical Report, RMS-9-95-2.

[7] Rus, D., Vona, M.: Crystalline Robots: Self-reconfiguration with Compressible Unit Modules. Autonomous Robots. January 2001. 10 (1): 107-124.

[8] Rus, D., Vona, M.: Self-Reconfiguration Planning with Compressible Unit Modules. Proceedings of the 1999 IEEE Int. Conference on Robotics and Automation. 2513-2520.

[9] Rus, D. Vona, M. A Physical Implementation of the Crystalline Robot, Proceedings of the 2000 IEEE Int. Conference of Robotics and Automation.

[10] McGray, C., Rus, D.: Self-Reconfigurable Molecule Robots as 3D Metamorphic Robots. Proceedings of the 1998 Conference on Intelligent Robot Systems. 1

[11] Suh, J.W., Yim, M. and Homans, S.B.: Telecubes: Mechanical Design of a Module for Self-Reconfigurable Robotics. Proceedings of the 2002 IEEE Int. Conf. on Robotics and Automation.

[12] Vassilvitskii, S., Kubica, J., Rieffel, E., Suh, J., Yim, M.: On the General Reconfiguration Problem for Expanding Cube Style Modular Robots. Proceedings of the 2002 IEEE Int. Conf. on Robotics and Automation.

[13] Unsal, C., Kiliccote, H., Patton, M., Khosla, P.: Motion Planning for a Modular Self-Reconfiguring Robotic System. Distributed Autonomous Robotic Systems 4, Springer, November, 2000.

[14] Walter, J., Welch, J. and Amato, N.: Distributed Reconfiguration of Metamorphic Robot Chains. Proceedings of the Nineteenth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2000), Portland, Oregon, 2000, pp. 171-180.

[15] Yoshida, E., Murata, S., Kaminura, A., Tomita, K., Korokawa, H. and Kokaji, S.: Motion Planning for a Self-Reconfigurable Modular Robot. Seventh International Symposium On Experimental Robotics, 2000.

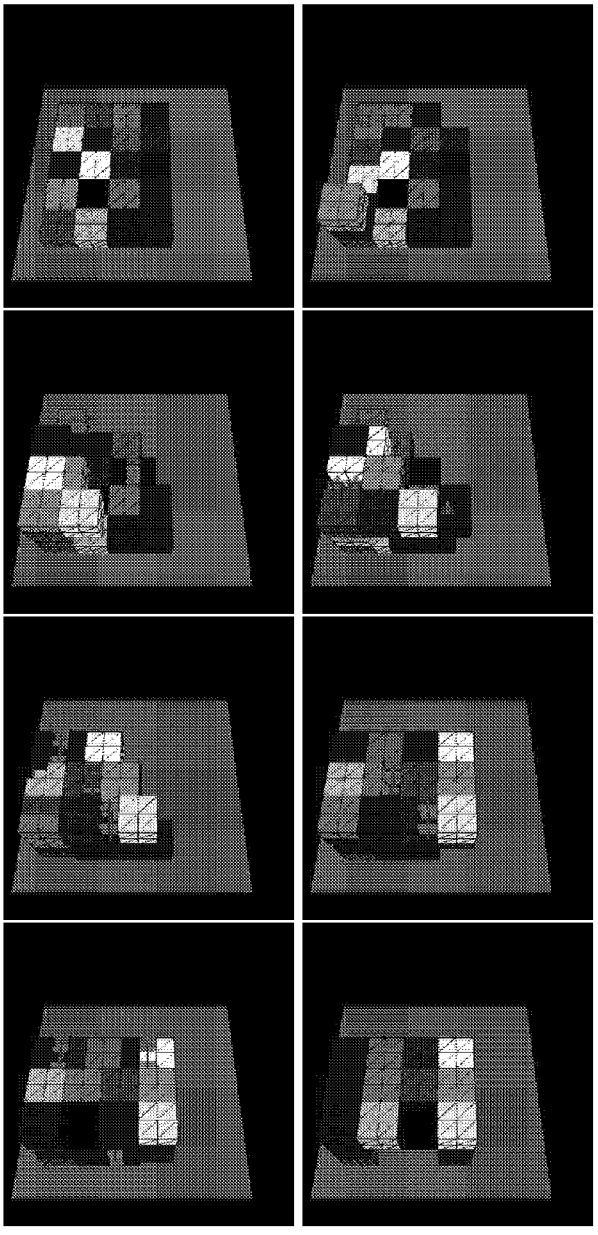

Figure 8: Snapshots of 311 step autonomous reconfiguration from flat sheet to a table with 20 meta-modules.