Self-reconfiguration Planning with Compressible Unit Modules

Daniela Rus Marsette Vona

Department of Computer Science
Dartmouth College
Hanover, NH 03755

{rus,mav}@cs.dartmouth.edu

Abstract

We discuss a robotic system composed of Crystalline *modules. Crystaline modules can aggregate together to form distributed robot systems. Crystalline modules can move relative to each other by expanding and contracting. This actuation mechanism permits automated shape metamorphosis. We describe the crystalline module concept and show the basic motions that enable a crystalline robot system to self-reconfigure. We present an algorithm for general self-reconfiguration and describe simulation experiments.*

1 Introduction

We wish to develop versatile and extensible massively-parallel distributed robotic systems. We believe that versatility can be obtained by developing self-reconfigurable systems. Self-reconfiguring robots have the ability to adapt to the operating environment and the required functionality by changing shape. They consist of a set of identical robotic modules that can autonomously and dynamically change their aggregate geometric structure to suit different locomotion, manipulation, and sensing tasks. For example, a self-reconfiguring robot system could self-organize as a snake shape to pass through a narrow tunnel and reorganize as a multi-legged walker upon exit to traverse rough terrain.

Self-reconfiguration poses new challenges to designing and controlling distributed robot systems. Because the connectivity topology of these systems changes dynamically, new models of synchronization, communication, control, and planning are needed. Solutions to these problems will impact more broadly computation in distributed systems. For example, new models of distributed computing are emerging due to the proliferation of wireless mobile computers. As with self-reconfiguring robot systems, topology and reachability changes dynamically in wireless networks of mobile computers, requiring new solutions to communication and routing[1].

In our previous work [13, 14, 18] we discuss a small and simple robotic module we call a *Robotic Molecule*. This module has already been prototyped. Our experiments demostrate that it is capable of self-reconfiguration in three dimensions. We have also demonstrated how systems composed of robotic molecules can use self-reconfiguration to increase their locomotive versatility [15, 16]. The robotic molecule uses 4 rotational degrees of freedom to accomplish motion relative to a structure that consists of identical modules.

In this paper we propose a different approach to self-reconfiguring robot systems. The new approach uses an actuation mechanism inspired by that that of muscles and amoebas. The idea is to create a mechanical module that can expand and contract by a constant factor. In addition, this module should be capable of making and breaking connections with neighboring modules. By expanding and contracting the neighbors in a connected structure, an individual module can be moved relative to the structure. This basic operation leads to new algorithms for global self-reconfiguration. For example, Figure 2 shows three snapshots from a simulation of a four-unit robot.

The module we propose is called the *Crystalline* module. It has cubic shape with connectors to other modules in the middle of each face. This module is activated by three binary actuators that permit the cube to shrink and expand by a factor of two. This actuation scheme allows an individual module to relocate to arbitrary positions on the surface of a convex structure of modules in constant time (unlike previous systems that required linear time in the number of modules on

[1]The solutions employed by cellular phone systems rely on global broadcast, which is inadequate for our applications.

the surface [31, 24, 15, 16]). In this paper we focus on algorithms for shape metamorphosis planning for the class of robot systems consisting of crystalline modules, called *crystals*. We present an algorithm called *melt-grow* for self-reconfiguration where the initial and the goal structures have the same volume. The melt-grow planner achieves shape morphing by using an intermediate structure, which is a projection of the robot modules into a pool on the ground. This approach leads to reconfiguration algorithms that maintain stability in real-world environments where gravity is present. The planner runs in $O(n^2)$ time, where n is the number of modules in the crystal. We also describe a simulator we have built to study crystalline robot systems and give examples from our experience using this simulator.

2 Related work

Related work in self-reconfiguring robots includes robots in which modules are reconfigurable using external intervention [5]. In [6] a cellular robotic system is proposed to coordinate a set of specialized modules. Several specialized modules and ways of composing them were proposed. [30] studies multiple modes of locomotion that are achieved by manually composing a few basic elements in different ways. This work also presents extensive examples of locomotion and reconfiguration in simulation. [19, 31, 29, 20] consider a system of modules that can achieve planar motion by walking over one another. The reconfiguration motion is actuated by varying the polarity of electromagnets that are embedded in each module. More recently [21] this group developed a twelve DOF module capable of three-dimensional motion. [24] describes metamorphic robots that can aggregate as two-dimensional structures with varying geometry. The modules are deformable hexagons. This work also examines theoretical bounds for planning the self-reconfiguring motion of such modules. In [18] we have shown a constant-time reduction between robotic molecule structures our group has designed to support self-reconfiguration [15, 16] and metamorphic robots [24].

The robot proposed in this paper is different than the previously proposed modules in its actuation capabilities, which lead to new types of self-reconfiguration planning algorithms. The high-level idea of a shrinkable module that can be a cell in a reconfigurable system has been presented as the patent [28].

3 The Crystalline Module

3.1 Concept

The idea behind the crystalline module is to create a mechanism that has some of the motive properties of muscles, that can be closely packed in 3D space, and that can attach itself to similar units. We chose a design based on cubes with connectors to other modules in the middle of each face. Our idea is to build a cube that can contract by a factor of two and expand to the original size (see Figures 2 and 8). We wish to effect compression along all three principal directions (e.g., x, y, z) individually or in parallel. We call the module an *atom*, and each connector a *bond*. Figure 1 shows a design for the mechanics of a two-dimensional (square) implementation of the atom. Our idea is to use complimentary rack and pinion mechanisms to implement the shrinking and expansion. In three dimensions, the rack and pinion mechanisms could be replaced with lead screws. We are currently constructing this design, which will be the subject of a different paper.

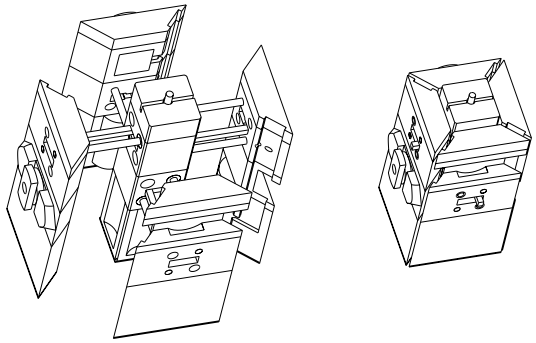

Figure 1: The mechanics of a 2D atom actuated by complimentary rack-and-pinion mechanisms. The atom is 4 inches tall (not including electronics, which are not shown). When expanded (left), the atom occupies a 4 inch square; when contracted (right) the atom occupies a 2 inch square.

3.2 Motion relative to a structure

A crystalline module can connect with identical modules to create crystalline robot systems. Only lattices whose faces are normal to the x, y, and z axes can be created using crystalline robots. By manipulating the size of the atom, it is possible to approximate any finite solid shape to an arbitrary precision using crystalline modules[2].

Crystalline robot systems are dynamic structures: (1) they can move using sequences of reconfigurations to implement locomotion gaits; and (2) they can undergo shape metamorphosis. The dynamic nature of

[2]The aliasing error for any shape on a raster display can be arbitrarily reduced by increasing the resolution of the display.

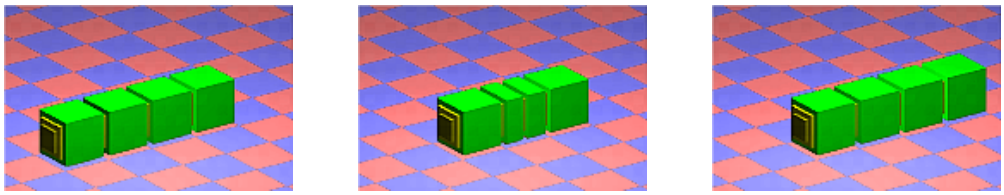

Figure 2: Three snapshots from a simulation using crystalline robots. The left image shows the initial state. The middle image shows the robot after shrinking two modules. The right image shows the robot after relaxing the shrunk modules. Notice that the entire structure moved forward one unit, in an inchworm-like fashion.

these systems is supported by the ability of individual modules to move globally relative to the structure.

The basic operations in a crystalline robot system are:

- `(expand <atom, dimension>)` - expand a compressed atom in the desired dimension (x, y, or z)
- `(contract <atom, dimension>)` - compress an expanded atom in the desired dimension
- `(bond <atom, dimension>)` - activate one of the atom's connectors to bond with a neighboring atom in the structure
- `(free <atom, dimension>)` - deactivate one of the atom's connectors to break a bond with a neighboring atom in the structure

Figure 2 illustrates the use of these primitives for generating a linear propagation algorithm called the *inchworm propagation* algorithm for crystals. The robot consists of four connected crystalline modules. The modules rest on a substrate of other crystalline modules[3]. We assume that each module can compress by a factor of 2. In the first phase of the algorithm, the rightmost module attaches to the substrate and the middle modules compress. This operation causes the leftmost module to advance by one unit (where the unit is denoted by the size of the module). In the second phase of the algorithm the leftmost module makes a connection to the substrate, the rightmost module disconnects and the middle two modules expand. The net effect of this algorithm is a global translation of one unit for the crystal. It is possible to describe similar algorithms for effecting global translations and 90 degree concave and convex transitions about crystalline structures.

[3]Note that the substrate is not necessary for all locomotion gaits.

3.3 Relocating a Module on a Crystal

An interesting property of these robots is illustrated in Figure 3. The problem is to find a way to move the crystalline module on the surface of the cubic crystal to any other location. Instead of propagating the module along the surface of the large cube, which would require time linear in the side length of the cube, it is possible to reach the goal using a constant number of primitive operations. The number of operations remains constant no matter where the start and goal locations are oriented relative to each other. First, the module is pulled inside the cube by compresing two internal modules, selected depending on the goal location. The two compressed modules are at the intersection of the supporting line for the starting location and the supporting line for the final location. Second, the compressed modules are expanded in the direction of the goal location so as to pop out a crystalline module in the desired place. If the two supporting lines do not intersect, two transitions will be required instead of one. Note that this popped module is not identical to the original module on the face of the structure. Using this algorithm, a module can be relocated in constant time on any convex substrate. This algorithm assumes that the actuators are strong enough to pull or push any number of modules during these two operations. If the actuators are not strong enough an inchworm-like propagation can be used instead, but this will no longer be a constant time operation. Alternatively, parallelism can be employed to retain constant-time performance.

When the crystalline robot structure is non-convex, a similar algorithm effects the module relocation operation in $O(k)$-time, where k is the number of concave angles in the structure. The idea is to iterate the algorithm for convex substrates at each concave angle between the starting location and the goal location.

Figure 4 shows the details of the algorithm for relocating a module on the surface of a crystal with two

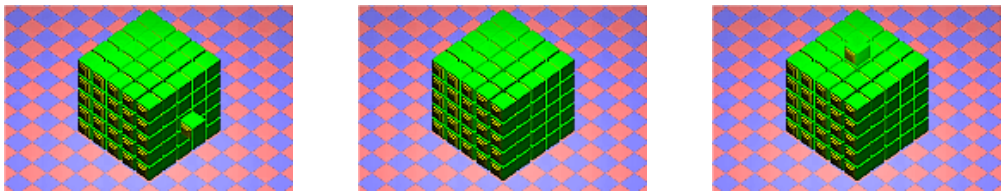

Figure 3: A crystalline module can be pushed into the large cube and popped out at any location on the surface of the cube in constant time. The three images are snapshots from a simulation. The left image shows the initial configuration (with the extra cube located on the side face) and the right image shows the final configuration (where the extra cube is on the top face). The middle image shows the base cube where two internal modules are compressed (not visible in the figure).

concave corners. The algorithm iterates compression and expansion steps.

We define a *scrunch* to be two adjacent, connected atoms that are compressed in the dimension normal to their connected faces. We define an *axis* to be a connected string of at least two atoms along one dimension. Two axes intersect if they have one atom in common. It follows that:

Theorem 1 *If an axis contains a scrunch, that scrunch can be moved to any position on the axis by the inchworm propagation algorithm. If one of two intersecting axes contains a scrunch, that scrunch can be transfered to the other axis, provided there exists sufficient surrounding structure to maintain connectedness throughout the operation.*

Proof: We have described the intuition behind these results. The technical proof is omitted for space considerations. □

To obtain a general algorithm for relocating crystalline modules on the surface of three-dimensional crystals the following two steps are sufficient:

1. Find a path with segments oriented along the x, y, and z directions from the starting location to the target location.

2. Iterate the algorithm in Figure 4 to travel around each turn in the path.

The complexity of the algorithm is $O(t)$, where t is the number of turns in the path. Note that $t = O(k)$, with k the number of concave angles in the structure.

4 A Planner for Shape Metamorphosis

In this section we describe a planner for self-reconfiguration in crystalline robot systems. More precisely, given a pair of crystals (S, G), each composed of n atoms, find a feasible reconfiguration plan P that transforms S into G. A reconfiguration plan P is a partially ordered sequence of primitive operations. A reconfiguration plan is feasible iff at no time during the execution of the plan does the crystal become disconnected.

The key observation for planning is that our crystalline systems consist of identical modules. Since all the modules are identical and interchangeable, it is not necessary to compute goal locations for each element. Thus, self-reconfiguration is different from the related warehouse problem (where modules are assigned unique ids and have to be placed at desired locations), which is intractable.

We have developed a centralized planning algorithm called the *melt-grow* planner that is complete over a useful subset *Grain(4)* of crystals and runs in $O(n^2)$ time, where n is the number of atoms in the crystal:

1. *Melt* S into an intermediate crystal I[4].

2. *Grow* G out of I.

The set *Grain(n)* contains the crystals that can be tiled by cubes of $n \times n \times n$ atoms (or squares of $n \times n$ atoms in 2D) called *grains*, so that the set of planes (or edges in 2D) that coincide with all sides of all grains intersect only at grain edges and corners. Figure 5 gives an example of a 2D crystal in Grain(4) and a 2D crystal not in Grain(4). The subset Grain(4) is useful because we can argue that by manipulating the scale of the atom, it is possible to approximate any finite solid shape to an arbitrary precision with a crystal in Grain(4).

[4] I is the projection of a 3D structure onto the ground plane or the projection of a 2D structure onto a line.

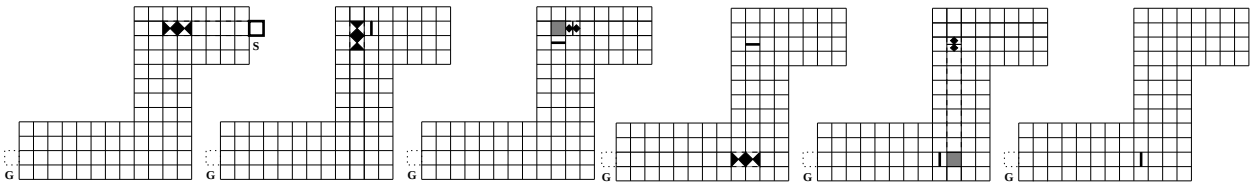

Figure 4: This figure illustrates the algorithm for relocating a crystalline module on a concave substrate of crystalline modules. The left figure shows the initial configuration. The relocating atom is in the upper right corner of the structure. The goal location is in the bottom left corner. Large dark diamonds mark two atoms about to be be compressed. Small dark diamonds mark two compressed atoms about to be expanded. Dark lines mark compressed pairs. The second figure shows the structure after the compression of the first pair of candidate atoms, and two atoms preparing for the next compression. The third figure shows two pairs of compressed atoms and a hole. The fourth figure shows the first compressed pair expanded into the hole and a candidate pair of atoms for the next compression. The fifth figure shows the state of the structure after this compression, with the resulting hole. The right-most figure shows the structure after an expansion into the hole. At this point, the remaining compressed pair can be expanded into the goal location.

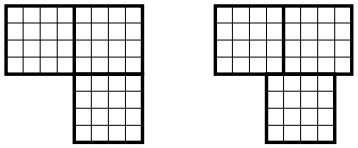

Figure 5: (Left): a 2D crystal in Grain(4). (Right): a 2D crystal not in Grain(4).

We use the intermetiate crystal I both to maintain stability during reconfiguration and in order to avoid backtracking[5]. We project the 3D structures S and G to and from a planar I. Physical stability can be guaranteed throughout the reconfiguration in environments where gravity is present by transforming S into I from the top down and I into G from the bottom up.

At a high level, the Melt algorithm works by finding a *mobile grain* g in S, transporting g to a place in I, and repeating until all grains are in I. Similarly, the Grow algorithm works by selecting mobile grains from I and transporting them to locations in G until all grains are in G. A grain is *mobile* iff it can be removed without disconnecting the crystal.

Figure 6 shows the details of the melt-grow algorithm. The first step in the algorithm is to compute the location of the intermediate structure I. This is done by the function `Locate-Stem`. The function

[5]Another planner might generate intermediate states in which no out-of-place grain is mobile, which would require backtracking.

```
Melt-Grow(Crystal S, Crystal G)
    Grain stem <- Locate-Stem(S)
    Crystal I <- Melt(S, stem)
    Grow(I, stem, G)

Melt(Crystal S, Grain stem)
    Crystal I <- stem
    S <- S-stem
    Crystal C <- Design-Pool(Volume(S), stem)
    While !empty(S)
        Grain mover <- Find-Mobile(S)
        Grain parent <- Find-Parent(I, C)
        Transport(mover, parent, union(S,I))
    return I

  Grow(Crystal I, Grain stem, Crystal G)
    Crystal C <- stem
    I <- I-stem
    While !empty(I)
        Grain mover <- Find-Mobile(I)
        Grain parent <- Find-Parent(C, G)
        Transport(mover, parent, union(C,I))
```

Figure 6: The implementation of the melt-grow algorithm.

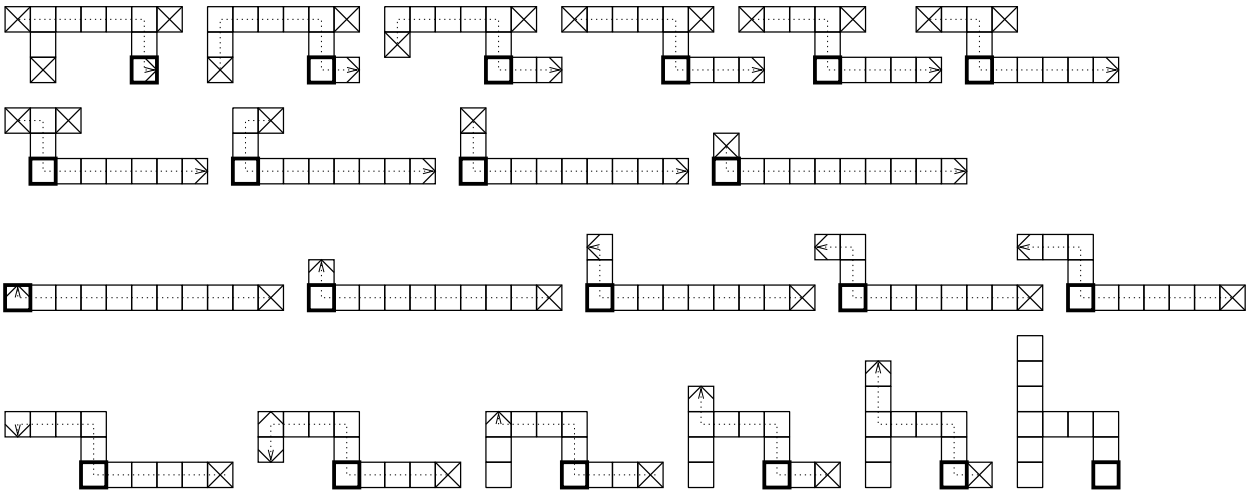

Figure 7: Reconfiguring a *Table* to a *Chair* using the melt-grow algorithm. Each square represents one grain (16 atoms). Grains marked ☒ are mobile, the stem grain is marked ☐, and candidate parent grains are marked ☐. This figure represents schematically the output of a simulation where the table and chair are composed of 176 crystals each.

`Design-Pool` generates an intermediate crystal with suitable volume. Many types of intermediate crystal are possible; for simplicity, we use a planar spiral of grains (or a one-dimensional line of grains for 2D systems). Such an intermediate crystal design makes locating mobile grains in I trivial.

The second step of the algorithm is to melt S into I; the third step is to grow I into G. These steps require locating a mobile grain, locating a good destination for this grain, and finding a path for the grain to the destination. Mobile grains in crystal C can be located by searching for vertices which are not articulation points in $GCG(C)$, the Grain Connectivity Graph of crystal C. $GCG(C)$ is an undirected graph whose vertices represent grains in C and whose edges represent active connections between neighboring grains. While Melting, any mobile grain still in S is a suitable mover; while Growing, any mobile grain still in I is a suitable mover. Parent grains can be located by searching for grains that are adjacent to yet-to-be-filled vacancies. While Melting, any such grain in I is a suitable parent; while Growing, any such grain in G is a suitable parent.

After locating a mobile grain and a parent, the mobile grain is transported to a space adjacent to the parent by (1) Finding a route through the crystal (i.e. by Depth-First-Search from parent in $GCG(C)$); (2) Decomposing the route into a sequence of grain motion primitives, and from there into a sequence of atom motion primitives; and (3) Executing the atom motion primitives. The following are the grain motion primitives:

- `(scrunch <grain, dimension, sense>)` - create a planar (linear in 2D) compression in a mobile grain at one of its six faces ($+x$, $-x$, $+y$, $-y$, $+z$, $-z$)
- `(relax <grain>)` - expand a compression at one face of a grain into a grain-in-progress
- `(transfer <grain>)` - transfer a compression at one face of a grain to the adjacent grain
- `(propagate <grain>)` - move a compression at one face of a grain to the opposing face of the grain
- `(convert <grain, dimension, sense>)` - move a compression at one face of a grain to one of the faces in an orthogonal dimension

Any mobile grain in a Grain(4) crystal can always be `scrunch`ed. A scrunch can always be transported to any parent grain by a sequence of transfer, propagate, and convert operations. Finally, any candidate parent containing a scrunch can always `relax` into an adjacent grain-in-progress. The algorithmic simplicity afforded by these guarantees is the reason for introducing the Grain(4) abstraction.

Figure 7 shows an example where the melt-grow algorithm was used to automatically transform a 2D *table* into a 2D *chair*. Since the melt-grow algorithm operates at the grain level, not the atom level, each square in Figure 7 represents one grain (16 atoms).

Theorem 2 *The melt-grow algorithm is complete for the subset Grain(4) of crystals. The running time of the melt-grow algorithm is $O(n^2)$, where n is the number of atoms in the crystal.*

Proof: Omitted for space considerations. □

5 Simulation

We have developed a simulator for 3D crystalline robots called *xtalsim*. We implemented the melt-grow algorithm in xtalsim, and we have performed experiments in simulation which verify that crystalline robots can self-reconfigure. The entire *table* to *chair* metamorphosis in Figure 7 was generated automatically by the melt-grow planner. Each object consists of 176 atoms. Figure 8 shows several snapshots from a smaller 3D simulation in which a dog-shaped object is made to metamorphose into a couch-shaped object via a hand-written plan.

Simulations are specified to xtalsim in a high-level language, either as initial and goal Grain(4) crystals, as sequences of grain motion primitives, or as sequences of atom motion primitives. The language includes constructs to allow parallel and serial combinations of actions and supports reusable, modular algorithms.

Xtalsim facilitates high-level experimentation with crystalline robots since it (1) efficiently extrapolates the global effects of individual atom movements, (2) verifies that all specified actions are feasible, and (3) displays the resulting atom actions as a 3D animation.

6 Discussion and Future Work

We presented the crystalline robotic module which is capable of self-reconfiguration by using a muscle-like actuation mechanism. The module is capable of making and breaking connections with identical modules, and of compressing by a constant factor. This actuation mechanism supports very fast algorithms for relocating one module on the surface of a crystal, which leads to an efficient $O(n^2)$ planner for shape metamorphosis. We are currently prototyping this module.

Acknowledgements

We are grateful to Keith Kotay for extensive discussions about the crystalline module. We are also grateful to Michael Shin for his help in implementing the simulator.

This paper describes research done in the Dartmouth Robotics Laboratory. Support for this work was provided through the NSF CAREER award IRI-9624286 and the NSF award IRI-9714332.

References

[1] Y. Cao, A. Fukunaga, A. Kahng, and F. Meng. Cooperative mobile robots: Antecedents and directions. Technical report, UCLA Department of Computer Science, 1995.

[2] I. Chen and J. Burdick. Enumerating the Non-Isomorphic Assembly Configurations of a Modular Robotic System. To appear in the *International Journal of Robotics Research*.

[3] P. Chew and K. Kedem. Getting around a lower bound for the minimum Hausdorff distance. In *Third Scandinavian Workshop on Algorithm Theory*, eds. O Nurmi and E. Ukkonen, Lecture Notes in Computer Science 621, pp 318–325, Springer Verlag 1992.

[4] G. Chirikjian and J. Burdick. Kinematics of a hyper-redundant robot locomotion with applications to grasping. In *Proceedings of the IEEE International Conference on Robotics and Automat ion*, 1991.

[5] R. Cohen, M. Lipton, M. Dai, and B. Benhabib. Conceptual design of a modular robot. In *Journal of Mechanical Design*, pp. 117-125, March 1992.

[6] T. Fukuda and Y. Kawauchi. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. In *Proceedings of the IEEE Conference on Robotics and Automation*, pp. 662-667, 1990.

[7] G. Hamlin and A. Sanderson. Tetrabot modular robotics: prototype and experiments. In *Proceedings of the IEEE/RSJ International Symposium of Robotics Research*, pp 390-395, Osaka, Japan, 1996.

[8] Kazuo Hosokawa, Isao Shimoyama, and Hirofumi Miura. Dynamics of self-assembling systems — analogy with chemical kinetics. *Artificial Life*, 1(4), 1995.

[9] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Matching and Machine Intelligence*, 1993.

[10] S. Kelly and R. Murray, Geometric phases and robotic locomotion. CDS Technical Report 94-014, California Institute of Technology, 1994.

[11] K. Kotay and D. Rus. Navigating 3d steel web structures with an inchworm robot. In *Proceedings of the 1996 International Conference on Intelligent Robots and Systems*, Osaka, 1996.

[12] K. Kotay and D. Rus. Task-reconfigurable robots: navigators and manipulators. In *The 1997 International Conference on Intelligent Robots and Systems*, 1997.

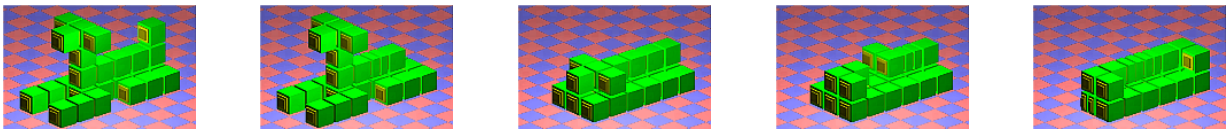

Figure 8: Five snapshots from a simulation using crystalline robots. The initial configuration (on the left) is a dog-shaped object. The final configuration (on the right) is a couch-shaped object. The middle images show intermediate steps in the transformation from dog to couch. The planning for this transformation was done manually. Note that some atoms are left in a compressed state so that the volume of the final shape is less than the volume of the initial shape.

[13] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfigurable robotic molecule. In *Proceedings of the 1998 International Conference on Robotics and Automation*, 1998.

[14] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule: design and control algorithms. In *the 1998 Workshop on Algorithmic Foundations of Robotics*, 1998.

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

[16] K. Kotay and D. Rus. Locomotion Versatility through Self-reconfiguration In *Robotics and Autonomous Systems*, 1998 (to appear).

[17] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers 1991.

[18] C. McGray and D. Rus. Motion Self-reconfiguring Molecules as 3D Metamorphic Squares In *Proceedings of the 1998 International Conference on Intelligent Robots and Systems*, 1998.

[19] S. Murata, H. Kurokawa, and Shigeru Kokaji. Self-assembling machine. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, 1994.

[20] S. Murata, H. Kurokawa, K. Tomita, and Shigeru Kokaji. Self-assembling method for mechanical structure. In *Artif. Life Robotics*, 1:111–115, 1997.

[21] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-D Self-Reconfigurable Structure. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, Leuven, 1998.

[22] R. Murray. Trajectory generation for underactuated systems with applications to robotic locomotion. In *Workshop on Algorithmic Foundations of Robotics*, eds. P. Agrawal, L. Kavraki, and M. Mason, A. K. Peters, 1998.

[23] B. Neville and A. Sanderson. Tetrabot family tree: modular synthesis of kinematic structures for parallel robotics. In *Proceedings of the IEEE/RSJ International Symposium of Robotics Research*, pp 382-390, Osaka, Japan, 1996.

[24] A. Pamecha, C-J. Chiang, D. Stein, and G. Chirikjian. Design and implementation of metamorphic robots. In *Proceedings of the 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference*, Irvine, CA 1996.

[25] C. Paredis and P. Khosla. Kinematic Design of Serial Link Manipulators from Task Specifications. In *International Journal of Robotic Research*, Vol. 12, No. 3, pp 274–287, 1993.

[26] C. Paredis and P. Khosla. Design of Modular Fault Tolerant Manipulators. In *The First Workshop on the Algorithmic Foundations of Robotics*, eds. K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, pp 371-383, 19 95.

[27] D. Rus. Self-Reconfiguring Robots. *IEEE Intelligent Systems*, 13(4), 2-5, July/August 1998

[28] K. Tanie and H. Maekawa. Self-reconfigurable cellular robotic system. US Patent 5361186, 1993.

[29] K. Tomita, S. Murata, E. Yoshida, H. Kurokawa, and S. Kokaji. Reconfiguration method for a distributed mechanical system. In *Distributed Autonomous Robotic Systems 2*, pp 17–25, Springer Verlag 1996.

[30] M. Yim. A reconfigurable modular robot with multiple modes of locomotion. In *Proceedings of the 1993 JSME Conference on Advanced Mechatronics*, Tokyo, Japan 1993.

[31] E. Yoshida, S. Murata, K. Tomita, H. Kurokawa, and S. Kokaji. Distributed Formation Control of a Modular Mechanical System. In *Proceedings of the 1997 International Conference on Intelligent Robots and Systems*, 1997.