A Basis for Self-Repair Robots Using Self-Reconfiguring Crystal Modules

Robert Fitch
Dept. of Computer Science
Dartmouth, USA

Daniela Rus
Dept. of Computer Science
Dartmouth, USA

Marsette Vona
A.I. Lab
MIT, USA

Abstract. Self-repair robots are modular robots that have the capability of detecting and recovering from failures. Typically, such robots are unit-modular and carry a number of redundant modules on their bodies. Self-repair consists of detecting the failure of a module, ejecting the bad module and replacing it with one of the extra modules. In this paper we show how self-repair can be accomplished by self-reconfiguring Crystalline robots. We describe the Crystalline robots, which consist of modules that can aggregate together to form distributed robot systems and are actuated by expanding and contracting each unit. This actuation mechanism permits automated shape metamorphosis. We also describe an algorithm that uses this actuation mechanism for self-repair.

1 Introduction

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. A primary design goal for a self-reconfiguring robot is to allow the robot to assume any geometric shape, as opposed to other types of shape-changing robots, which may only take one of a small number of shapes.

Self-reconfiguring robots typically consist of multiple modules, which confers the robot system redundancy. This built-in redundancy leads to interesting fault-tolerance and self-repair properties. If an arbitrary part of a fixed-architecture robot fails, the robot cannot usually perform self-repair; a human or a different robot must perform the task. Intuitively, a self-repair system must have at least two qualities: the ability to self-modify, and the availability of new parts or resources to fix broken parts. Most extant systems lack these properties. However, self-repair behavior is prevalent in biological systems, the most notable being human tissue repair [6], and this serves as inspiration for our work.

Our goal is to build on the ground-breaking work of Yoshida et al [19], who introduced the concept of a self-repair robot and presented a simulated-annealing algorithm for this operation. We focus instead on geometric motion planning algorithms for self-repairing robots consisting of Crystalline modules [12]. We begin by describing our approach to homogeneous[1] self-reconfiguring robot systems, which uses a module called a *Crystalline Atom* that is actuated by expansion and contraction. By expanding

[1]There are two basic types of self-reconfiguring robot systems: *heterogeneous* and *homogeneous*. In a heterogeneous system, the modules may be different. In a homogeneous system all the modules are identical.

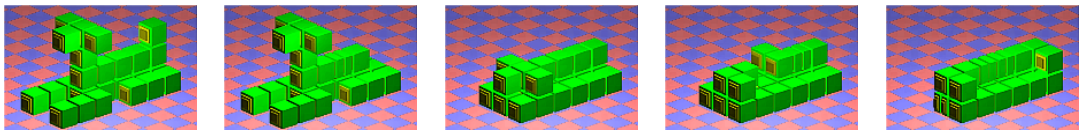

Figure 1: Five snapshots from a Crystalline robot simulation. The initial configuration is a dog-shaped object, and the final configuration is couch-shaped. The middle images show intermediate steps in the transformation. The planning for this transformation was done manually. Note that some atoms are left in a compressed state.

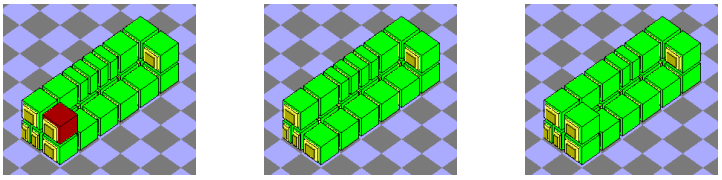

Figure 2: The left image shows the couch in Figure 1 with a defective module highlighted. The middle figure shows that couch after the bad module was ejected. The right figure shows the repaired couch. One of the compressed modules was used to fill in the gap.

and contracting the neighbors in a connected structure, an individual module can be moved in general ways relative to the entire structure. This basic operation leads to new algorithms for global self-reconfiguration planning. This basic operation allows a crystalline robot system to realize a wide range of geometries; for example, Figure 1 shows snapshots from a simulation in which a dog-shaped object transforms itself into a couch-shaped object. We focus on algorithms that permit a robot to detect a failed module, eject it, and replace it with one of the extra modules on the body, so as to repair the arm-rest of the couch in Figure 2, for instance.

2 Related Work

This research was done in the context of previous work in self-reconfiguring and self-repair robots. Yoshida et al [19] present the first algorithm for self-repair in self-reconfiguring robots, based on simulated annealing. Fukuda et al propose a cellular robotic system to coordinate a set of specialized modules [1]. Several specialized modules and ways of composing them were proposed. Yim studies multiple modes of locomotion that are achieved physically by manually composing a few basic elements in different ways [17]. In [18], Yim proposes a dodecahedron-based module capable of self-reconfiguration. In [8, 15, 9], Murata et al consider a system of modules called *Fracta* that can achieve planar motion by walking over one another, with actuation provided by varying the polarity of embedded electromagnets, and generalize to 3-D motion in [10]. In [11] Chirikjian et al describe metamorphic robots that can aggregate as 2-D structures with varying geometry, and examine theoretical bounds for self-reconfiguration planning. In [7] we have shown a constant-time reduction between robotic molecule structures our group has designed to support self-reconfiguration [2, 3] and metamorphic robots [11]. In [12, 13] we described the Crystalline robot. This robot's novel compression/expansion actuation leads to new types of self-reconfiguration planning algorithms. The high-level idea of a shrinkable module in a reconfigurable system has

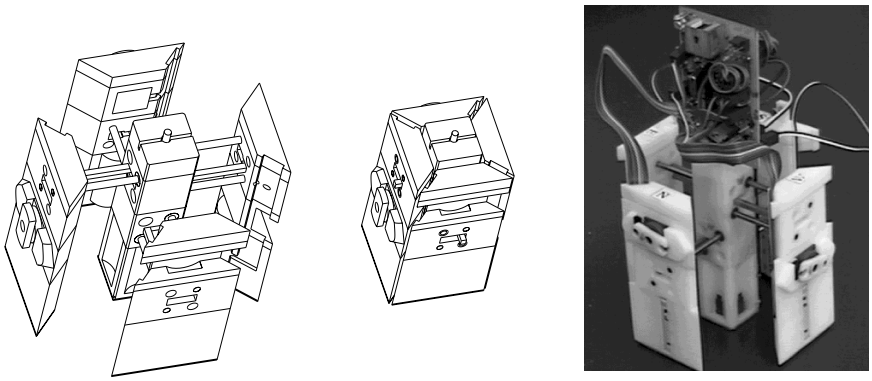

Figure 3: The mechanics of a 2D Atom actuated by complimentary rack-and-pinion mechanisms. The Atom is 4 inches tall (not including electronics, which are not shown). When expanded (left), the Atom occupies a 4 inch square; when contracted (middle) the Atom occupies a 2 inch square. The physical prototype for the Crystalline Atom is shown at the right.

been presented by Tanie et al as the patent [14].

3 The Crystalline Module

3.1 Concept

The Crystalline module is a mechanism that has some of the motive properties of muscles, can be closely packed in 3-D space, and can attach itself to similar units. We chose a design based on cubes with connectors to other modules in the middle of each face. The idea is to build a cube that can contract by a factor of two and expand to the original size (Figure 1). 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 3 shows a design for the mechanics of a 2-D (square rather than cubic) implementation of the Atom and also the physical prototype. We use complimentary rack and pinion mechanisms to implement the contraction and expansion actuation for the 2-D prototype. In three dimensions, the rack and pinion mechanisms could be replaced with lead screws. Because squares and cubes are highly regular, most planning algorithms developed in two dimensions can be easily generalized to three. Similarly, algorithms developed in three dimensions can be easily specialized to 2-D structures.

3.2 A Physical Implementation

Crystalline Atomic modules can be constructed in both two and three dimensions. In two dimensions, Atoms are square; in three dimensions they are cubic. In this section we describe the 2-D version of the module we constructed, shown in Figure 3. The module has an expansion/contraction ratio of 2. All the faces of the Atom have to be fully extended or fully contracted. Each face of the module has a connection slot. However, only two out of the four faces[2] have active connection slots (see Figure 3). These active

[2]The active connection slots are situated on adjacent faces, which allows any lattice of Crystalline Atoms to be fully connected.

slots provide a key-and-lock mechanism for forming rigid connections with adjacent modules. Thus, the entire unit can be realized with three degrees of freedom: one to expand/contract the faces of the Atom, and two for the active connectors. All three degrees of freedom can be implemented with binary actuators. Since Atoms can never rotate relative to one another, the use of two rather than four connectivity degrees of freedom leads to no mechanical limitations. Every inter-Atomic interface of a structure will have one active connection mechanism. The module has on-board electronics and four 3V 2/3A size Lithium batteries, so that it can function untethered.

The expansion/contraction degree of freedom has been implemented with a rack-and-pinion mechanism. A vertically mounted pinion at the center of the core mates all rack simultaneously. Gear racks are rigidly mounted to the rear of each face. Racks from opposing faces are mounted off-center and racks from adjacent faces are staggered vertically so that they do not overlap at the center of the core. The pinion is driven by a gear motor. Spinning it in one direction extends the racks, which causes the Atom to expand. Spinning in the other direction retracts the racks which causes the Atom to contract. The motor used for these motions is a Lego toy *Mini-Motor*, which was selected for its small size (5/8 inch cube), useful torque (2oz-in at 10rpm, 12V and 80mA), and low-cost ($11). Two hall-effect sensors serve as bi-level position sensors and are used to control the expansion/contraction movements of the Atoms. One sensor is used to determine if the faces have moved close enough to the core during a contraction movement; the other sensor indicates whether the faces have moved far enough outward during an expansion.

The connection mechanisms are based on a *channel and key* concept. The passive face contains a deep horizontal channel on its outer surface. Pockets are built into the upper and lower inside surfaces of this channel at the center of the face. The active face contains a gear-motor (the same model Lego mini-motor used to actuate expansion/contraction). A bar is attached to the output shaft of the motor. At one angle, the bar can move unobstructed through the channel of the passive face and the connector is freed. At another angle, the bar is rotated so that it extends into the pockets of the passive face and the connector is bonded.

When fully contracted, the Atom is a square with a 2 inch side. When fully expanded, the Atom is a square with a 4 inch side. The height of the Atom is 7 inches and its weight is 12 ounces. A Crystalline Atom 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[3].

Each Atom contains an on-board processor (Amtel AT89C2051 microcontroller), power supply (four 2/3 A Lithium batteries), and support circuitry, which allows both fully untethered and tethered operations. Atoms are connected by a wired serial link to a host computer to download programs. For untethered operations, an experiment specific operating program specified as a state sequence is first downloaded over a tether. When the tether is removed, an on-board IR receiver is used to detect synchronization beacons from the host.

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

4 Self-Repair

In this section we develop an algorithm that allows systems of Crystalline modules to self-repair. The main benefit of this capability is an increase in the reliability of our robots. Our work is inspired by Murata et al[19].

We start by observing that the process of self-repair consists of three phases: (1) detect failure, (2) eject the failed module, and (3) replace the failed module. We have not yet addressed detecting module failure, as in general it is highly dependent on the implementation of the system. There are a number of possible approaches, however, such as polling by a central unit or nearest-neighbor testing. Biological systems, such as human skin cells, use a "cry for help" method where the failed unit (damaged cell) sends a broadcast message by releasing its contents into the microenvironment as it dies[6].

We present algorithms for phases (2) and (3) of self-repair. Our solution assumes Crystal robots that can self-reconfigure in the plane only, with a known failed module. Without loss of generality, we focus our discussion on the case when only one module fails in the system. Our algorithms can be iterated to cope with multiple module failures.

4.1 Ejecting a bad module from the system

A bad module may not be able to move under its own power, so "live" modules in the system should manipulate the "dead" module into position for ejection. The strategy we pursue is to move the dead module to a place where it will simply fall off of the robot when released by any attached live modules. Our solution (1) identifies all the location on the surface of the robot from where it is possible to release the bad module; (2) computes a shortest path to that region; and (3) uses a gait to propel the bad module along the shorted path.

Since we envision robots comprising large numbers of modules, we are concerned with scalability. Therefore, we would like an algorithm that is parameterized by the overall geometric shape of the robot rather than the number of individual modules. Assuming such a model, which is simple polygon with holes, we refer to the locations we compute for ejection as *cliff edges* (see Figure 4). Intuitively, a cliff edge is an edge that would be lit by a light source along the x-axis, if the robot is placed in the first quadrant.

We developed an algorithm called *find-cliffs* that analyzes a geometric representation of a crystalline robot and returns a set of cliff edges. Intuitively, we choose outer edges of the robot that can "see" the ground. Assuming the robot is not embedded in the surface on which it sits, there are unique bottom-left and bottom-right vertices. Considering the right half of the robot, we begin at the bottom-right point and walk through the edges. As long as we walk toward non-decreasing x values, the edges are cliff edges. If we reach a non-cliff edge, however, there are two cases to consider. First, the edge may point up and in towards the structure (increasing y, decreasing x). Here we simply draw an imaginary vertical line and continue the edge walk until we intersect this line, where we proceed as before. If the edge angles down and towards the robot (decreasing y, decreasing x), then we have mislabeled edges as cliff edges. We skip this edge and continue, and make a second pass from top to bottom to delete the incorrectly labeled cliff edges. The other side of the robot is processed symmetrically.

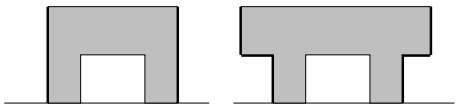

Figure 4: A simple polygone denoting a robot, with highlighted cliff edges.

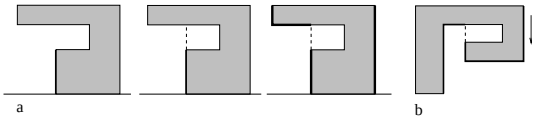

Figure 5: Two cases for non-cliff edges. (a) Jumping to intersection point. (b) Second pass deletes mistakes.

Algorithm 1

1. Determine the bottom-right, right-most, bottom-left, and left-most vertices v_0, v_1, v_2, v_3.
2. Walk through edges from v_0 to v_1, choosing edges that angle away from robot.
3. If the next edge angles up and towards the robot, jump vertically and continue. Else if the edge angles beneath previously marked cliff edges, skip and continue.
4. Make a second pass, from v_1 to v_0, deleting mislabelled cliff edges.
5. Repeat for v_2 to v_3, symmetrically.

Due to space considerations, we omit the correctness proof for this algorithm.

4.2 A Planner that Minimizes Turns

Given a robot, a failed module, and a set of cliff edges, we would like to have an efficient algorithm for moving the failed module to the cliff edge and replacing it with a spare. In this section we present an algorithm that solves both problems.

Recall that Crystal modules can only move rectilinearly in a robot structure. In a Crystal robot system, it is simpler and cheaper to move along a line than to execute a turn. Therefore, we focus our efforts on finding a motion plan that produces Manhattan paths and minimizes turns. The simplest solution is to use breadth-first search on the module-connectivity graph. However, the running time would then depend on the total number of modules, which we avoid for scalability and instead use the geometric model. If we parameterize the cost function of straight line motion and turns, we can use a shortest path algorithm to compute a path, and then iterate the appropriate gaits to traverse the path.

Finding minimum distance paths in Configuration space using the euclidean distance metric is well-studied [4]. Finding shortest paths in using the rectilinear metric has also been studied, mainly in the context of VLSI chip design. Lee et al [5] provide a useful review of problem flavors and solutions. Most notably, Lee et al define the Minimum Bend Shortest Path (MBSP) Problem and the Shortest Minimum Bend Path (SMBP) Problem. MBSP searches for a path with minimum bends, or turns, among all shortest rectilinear distance paths. SMBP searches for the minimum rectilinear distance path among all minimum bend paths. Algorithms to solve these problems use graph theoretic or wave front approaches (or both) and achieve the optimal running time of $O(eloge)$, where e is the number of obstacle edges. The algorithm we choose to use comes from Yang et al. [16], and uses a combination of the two approaches to solve both SMBP and MBSP.

Assuming the dead module can not move under its own power, we can select a gait to manipulate it to a cliff edge. One such gait is shown in figure 6. We use a MBSP algorithm for this particular gait, since straight line motion is easier to plan than turns but the gait actually requires constant time per unit distance. Once the dead module

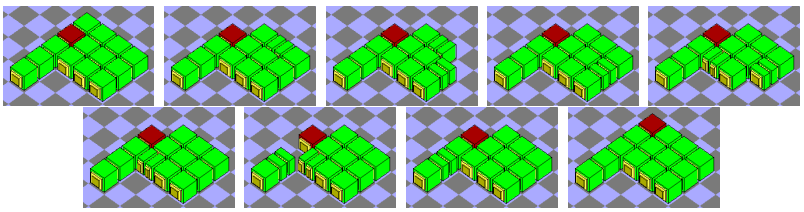

Figure 6: Simulation displaying sample gait to push dead module.

is ejected, we use a SMBP to plan the replacement motion, since module relocation is linear in the number of turns in the path.

The planning algorithm is summarized as follows:

Algorithm 2

1. Using the MBSP algorithm, compute a path from the dead unit to a cliff edge.
2. Iterate the pushing gait for each step in the path, eventually ejecting the module.
3. Compute the SMBP from the ejection location to the closest redundant module.
4. Use fast module relocation, following the computed path, to virtually relocate the replacement module to the ejection location.

5 Experiments in Simulation

We conducted self-repair experiments using the *xtalsim* simulator. Xtalsim was created as part of the original Crystalline robot development effort and was used to explore the self-reconfiguring abilities of the robots. The simulator reads a file-based specification of a robot and motions of individual modules. It verifies the validity of the requested motions and displays a 3-D animation of the robot in motion.

The existing xtalsim version required modification to support self-repair functionality. We extended the software to allow specification of dead modules along with graphical distinction in the animation. Also, we added removal features to simulate ejected modules. Figure 6 depicts snapshots from the simulator output. Our experiments demonstrate pushing gaits to move a disabled module, and module relocation to fill holes. We traverse the computed shortest path by iterating the pushing gait algorithms for straight line movement and for turns.

6 Conclusions and Future Work

We discussed the Crystalline robot and proposed a multi-step strategy for implementing self-repair. Our algorithms for ejecting and replacing failed modules analyze the robot structure and compute efficient motion plans, taking into account manhattan distance and turn minimization. Although here we describe self-repair in simulation, we are currently developing hardware experiments. Working with hardware will help us in addressing the problem of failure detection. We would also like to extend our algorithms from 2-D to 3-D models, and are considering issues such as planning for robots in motion and the idea of self-reconfiguration planning for more efficient module ejection.

Acknowledgements

We are grateful to Laura Ray and 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, NSF award IRI-9714332, NSF award EIA-9901589, and NSF award IIS-98-18299. We are grateful for it.

References

[1] 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.

[2] 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.

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

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

[5] D.T. Lee, C.D. Yang, and C.K. Wong. Rectilinear paths among rectilinear obstacles. Discrete Applied Mathematics, Vol. 70, pp. 185-215, 1996.

[6] G. Majno and I. Joris, *Cells, Tissues, and Disease: Principles of General Pathology.* Blackwell Science 1996.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] D. Rus and M. Vona. Self-Reconfiguration Planning with Unit Compressible Modules. *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, pp 2513–2520, Detroit, MI, 1999.

[13] D. Rus and M. Vona. A Physical Implementation of the Crystalline Robot. Submitted to the *2000 IEEE International Conference on Robotics and Automation*, San Frncisco, CA, 2000.

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

[15] 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.

[16] C.D. Yang, D.T. Lee, and C.K. Wong. Rectilinear path problems among rectilinear obstacles revisited. SIAM Journal of Computing, Vol. 24, pp. 457-472, 1995.

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

[18] M. Yim. Polypod II. `http://www.parc.xerox.com/spl/members/yim/`

[19] E. Yoshida, S. Murata, K. Tomita, H. Kurokawa, and S. Kokaji An experimental study on a self-repairing modular machine. *Robotics and Autonomous Systems*, Vol. 29, pp. 79-89, 1999.