

# Experiments in Distributed Control for Modular Robots

Zack Butler, Robert Fitch, and Daniela Rus  
Department of Computer Science  
Dartmouth College  
Hanover NH 03755, USA  
`{zackb,rfitch,rus}@cs.dartmouth.edu`

## Abstract

Effective algorithms for modular self-reconfiguring robots should be distributed and parallel. In previous work, we explored general algorithms for locomotion and self-replication and explained their instantiations to systems in which modules move over the surface of the robot. In this work, we present several algorithms and implementations applied to the Crystal robot: distributed locomotion algorithms designed specifically for unit-compressible actuation, as well as adaptation of a generic division algorithm to the Crystal robot. We also present the integration of a locomotion algorithm with a distributed goal recognition algorithm developed previously.

## 1 Introduction

Self-reconfiguring modular robots have the ability to reshape themselves into a wide variety of configurations to accomplish diverse tasks, such as locomotion through small passages, climbing tall obstacles, and moving while supporting large payloads. These robots usually comprise a large number of independent modules, each equipped with on-board processing, forming a distributed system. There are three key features that make such robots useful for operations in unknown environments: the ability to operate under uncertainty, the ability to move on unknown terrain and versatility and adaptation to multiple tasks.

Several types of modules and actuation mechanisms that can support self-reconfiguration have been proposed [4, 5, 7, 8, 13, 10]. In our previous work we described a module capable of 3D self-reconfiguration by using a rotation-based actuation called the *robotic molecule* [6] and a module capable of self-reconfiguration that uses scaling-based actuation called the Crystal [9]. The Crystal robot has gone through two design phases. The original design was described in [9]. Based on our experience with this module we refined the

design to add an additional degree of freedom, sensing, and distributed systems support through point to point communication.

The key algorithmic question for self-reconfigurable robots is the planning and control problem: how should the modules move relative to each other in order to achieve a static or dynamic goal shape, or to move in a desired way, and how to do it efficiently. We have already developed several centralized planners for self-reconfiguring robots [6, 9]. Some of the most interesting applications of this work will employ thousands of modules working together. The off-line planning algorithms proposed above move one module at a time and may be too slow and impractical for controlling lattices made of thousands of modules. In this paper we discuss distributed control algorithms that are scalable, support parallelism, and are better suited for operation in unstructured environments.

Distributed algorithms are naturally suited for controlling self-reconfiguring robots because they take advantage of modularity, allowing the system to be more robust to failures of individual modules and communications, and supporting partitioning of the robot. Several distributed algorithms for self-reconfiguring robots have been proposed, including locomotion for string-type robots [11] as well as reconfiguration of 2D and 3D robots. Notable 2D examples are algorithms for the Fracta system [12] and the system of Hosokawa *et al.* [5], and the PacMan algorithm for unit-compressible systems [1]. Algorithms for 3D systems include work on the Proteo system [15] as well as for meta-modules made from unit-compressible modules [14]. Many of these works have also included hardware implementation.

In our recent work [2, 3] we proposed distributed algorithms for several tasks: (1) *locomotion*, where the modular robot can implement a tumbling gait that conforms to the terrain geometry; (2) *self-replication*, where the robot can divide itself into smaller autonomous robots, for example to explore the terrain in parallel, and (3) *merging*, where two autonomous modular robots can connect into a larger robot, for example in order to climb taller obstacles. These algorithms use local information only and are inspired by a cellular automata approach. For each task we designed a set of geometric rules, so that each module tests the same rules with respect to its neighborhood to decide what action to take. The resulting control algorithms are distributed, efficient, and provably correct [2, 3]. These algorithms are also generic in the sense that they apply to an abstract model for self-reconfiguring robots, where individual modules on the robot have the ability to traverse a planar surface composed of identical modules and to make convex and concave transitions between surfaces. Most existing self-reconfiguring robots [5, 6, 7, 8, 13] fit this model.

In this paper we start from the algorithms developed in [2, 3] and show how they can be instantiated to unit-compressible systems such as the Crystal robot. This instantiation is challenging because the actuation mechanism of unit-compressible robots more naturally supports movement through the volume of the robot rather than on its surface. We discuss in detail the control algorithms for distributed robot locomotion and division. Each module can sense its local neighborhood structure, communicate with its neighbors, and perform some simple computations to evaluate the control rules. We also dis-

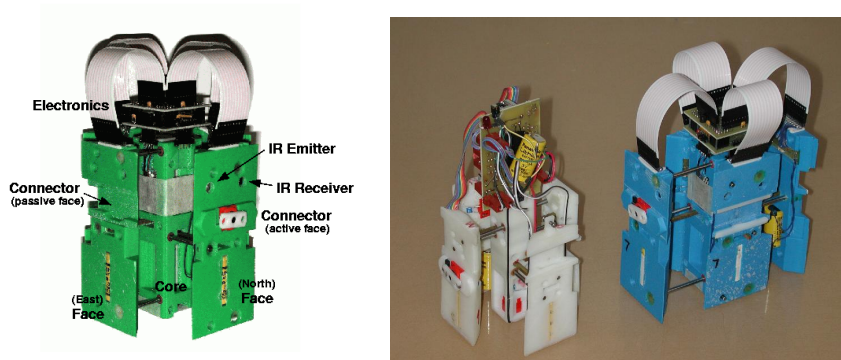


Figure 1: A single module of the Crystal robot (left), and together with the initial version (right).

cuss an implementation of these algorithms on the Crystal robot and present data from extensive hardware experiments performed with a 15-module Crystal robot.

## 2 The Crystalline Robot

The Crystalline Atomic Robot (or Crystal) is a unit-compressible self-reconfiguring modular robot. It actuates by expansion and contraction of individual modules, which together with connection and disconnection allows the robot to change shape as well as locomote. Each module consists of a central core and four faces that move in and out relative to the core to perform expansion. An expanded module is exactly twice the size of a compressed module, which aids in reconfiguration and planning.

The original version of the Crystal [9] had a single degree of freedom for expansion, so that all four faces expanded and contracted together. Each module had its own processor to control actuation, but synchronization was performed through sensing an external beacon — modules had no ability to communicate directly with each other.

The second-generation Crystal module, shown in Fig. 1, incorporates several important new features including an additional degree of freedom for actuation, inter-module IR communication capability and sensor inputs. A central control beacon is no longer present to coordinate behavior. Thus, the robot control must now be done in a purely distributed fashion. The North/South and East/West faces are independently actuated, so the degrees of freedom increase to four (two for expansion/contraction, and two for connectors on active faces). In addition, several features have been improved over the first version, including stiffening of the linear bearings that align the faces during actuation, more

powerful motors to perform actuation, and a faster processor with more memory and I/O capability.

Communication is implemented with asynchronous serial over IR components on the Crystal faces. Each module face contains an IR emitter and detector that allow modules to communicate at distances of up to 10 cm. These components are connected to a dedicated Maxim Max3100 UART in the core, so a unit can talk with all neighboring units essentially simultaneously. The UARTs communicate at 1200 Baud and have an eight-word hardware FIFO. Synchronous serial communication is used between the processor and the UARTs. On top of this hardware, we have developed a simple message-passing infrastructure, under which two-byte messages can be sent from one module to any of its neighbors. This allows us to write our control algorithms as message loops, triggering on incoming information about the state of the robot.

Dimensionally, this Crystal prototype is slightly larger than its predecessor. Expanded size is 5.2 inches square, and contracted size is 2.6 inches square. Overall height is 7.4 inches with a weight of 18 ounces. Fifteen modules have been constructed so far.

### 3 Experiments in Locomotion

One of the fundamental tasks of any robot system is locomotion. On most lattice-based systems, locomotion can be performed by having individual modules move over the surface of the group from the back to the front in a tank-tread-like pattern. Distributed algorithms that produce this type of motion were developed in [2]. In unit-compressible systems such as the Crystal, however, no single module can move relative to the group without help from other modules, and so a different specialized technique is required. Here we present two distributed algorithms which achieve locomotion with only local information available to each module.

#### 3.1 Locomotion Algorithms

Our locomotion algorithms (as well as the division algorithm described in Sec. 4.1) are based on a set of rules that test the module's relative geometry and generate expansions and contractions as well as messages that modules send to their neighbors. When a module receives a message from a neighbor indicating a change of state, it tests the neighborhood against all the rules, and if any rule applies, executes the commands associated with the rule. Both algorithms are designed to mimic inchworm-like locomotion: compressions are created and propagated from the back of the group to the front, producing overall forward motion.

The Stand-Alone algorithm is the simpler locomotion algorithm, and is presented as Algorithm 1. The overall idea behind Stand-Alone locomotion is that at any given moment, the majority of the modules are stationary. The remaining modules will move relative to the majority due to friction. The motion is

---

**Algorithm 1** Distributed Stand-Alone locomotion.

---

State: <i>nbr[]</i> , array of neighbors <i>heading</i> , direction robot is moving: N,S,E,W	Procedures: TryRules() position $\leftarrow$ FindPosition() <b>if</b> position = head <b>then</b> <b>if</b> <i>nbr</i> [opp( <i>heading</i> )] contracted <b>then</b> contract, send <i>state</i> expand, send <i>state</i> send <i>inch</i> <b>if</b> position = body <b>then</b> <b>if</b> <i>nbr</i> [opp( <i>heading</i> )] contracted <b>then</b> contract, send <i>state</i> expand, send <i>state</i> <b>if</b> position = tail and received <i>inch</i> <b>then</b> contract, send <i>state</i> expand
Messages: <i>inch</i> (direction <i>d</i> ), sent to move robot in direction <i>d</i> Action: set <i>heading</i> state to <i>d</i> , execute TryRules() <i>state</i> (state <i>s</i> ), announces state changes to neighbors Action: execute TryRules()	

---

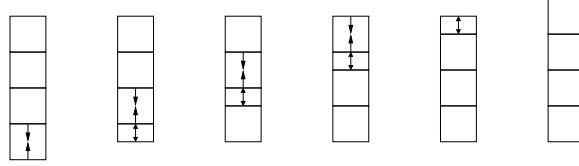


Figure 2: Schematic of module action under Stand-Alone locomotion, in which the group is heading upward. The series (left to right) represents progress of a single inchworm “step”, traveling half of a module width.

specified such that two adjacent modules will move synchronously, reducing the net force to the other modules. A schematic storyboard of this algorithm is given in Fig. 2. The “tail” module contracts first and sends a message to its forward neighbor with its new state. This module will then contract and send a message forward. Each module expands after contraction, so that the contraction propagates through the robot. When the contraction has reached the front of the group, the group will have moved half a unit forward (in theory; empirical results are given in Sec. 3.4). Once the leader of the group has contracted and expanded, it may then send a message back to the tail to initiate another step.

In contrast, the second algorithm, which we call Attaching locomotion, is more efficient in that more than one module can be contracted at one time. In addition, it can be successfully performed by as few as two modules, and is guaranteed to achieve the theoretical performance in terms of distance traveled per actuation cycle. However, the modules must attach to a row of fixed modules or have some other way to firmly fix themselves to the environment. The basic idea is that instead of a single contraction propagating from the tail to the head,

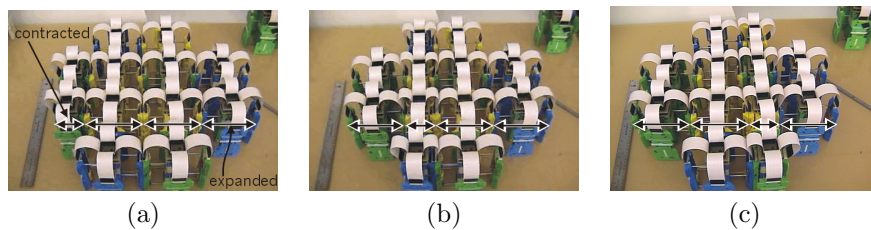


Figure 3: Photos of locomotion experiment for the blob shape mentioned in the bottom row of Table 1. In (a), the leftmost column is contracted, and in (b) and (c) the following columns contract to make the group walk to the right.

many contractions can exist at the same time as long as there is one fixed atom between each contraction.

### 3.2 Analysis/Extensions

Both locomotion algorithms can be proven correct, i.e. that they produce locomotion in the intended direction. This is done by noting that only the tail can contract at first, followed by each other module in turn. Since each contraction must be triggered by a *state* message, no module will contract until it has the proper information, but once it does contract and sends a message forward to that effect, the contraction will always propagate.

Algorithm 1 is specified for a single column, but can be extended to convex shapes by selecting one column as a master column. When a module in the master column actuates, a message is passed across its row that causes all modules in the row to actuate simultaneously. This is effective since communication is much faster than actuation, and the modules not in the master column have no other responsibilities that could cause communications lag. This allows for correct locomotion for any convex shape, as shown in Fig 3.

### 3.3 Hardware Instantiation





The ability to perform distributed locomotion depends on modules passing their state to their immediate neighbors. The communication infrastructure allows us to define a *state* message, which indicates whether the module is expanded in the direction of motion and (for the Attaching algorithm) whether it is connected to a fixed module. This information along with the message type easily fits within the two-byte limit. The other message required is the *inch* message, which tells the robot to begin locomotion, and includes the desired direction of travel. The *inch* message is sent from an external source to initiate the locomotion, and is also sent by the head module to trigger another step when desired. Together with a two message soft-boot sequence, these algorithms use a total of four message types to perform locomotion.

### 3.4 Results

Our first set of experiments attempted to empirically determine the effectiveness of the Stand-Alone locomotion algorithm. We tested single-row inchworms on different surfaces as well as more complex shapes (see Fig. 3). We found that with at least four modules in each row, the locomotion proceeds well, although its exact amount of progress is dependent on both the surface and the individual modules in the group.

In all experimental setups, we found that along fairly smooth surfaces (plexiglass and painted metal) the actual distance was 80-90% of theoretical for five-module rows, and nearly as good for four-module rows. These results held for both single- and double-column groups, as can be seen in Table 1. The bottom two rows in this table present an interesting phenomenon. The top and bottom rows added in the lower shape are only two modules long, and in fact work against the overall locomotion, so that this robot walked much more slowly than the rectangular shape in the previous row.

Table 1: Stand-Alone locomotion performance.

Robot shape	Surface	$N_t$	$N_s$	Mean distance (in/step)	Min-Max distance (in/step)
	Paint	10	10	2.24	2.10-2.39
	Plexiglass	10	10	2.32	2.22-2.37
	Plexiglass	10	6	2.14	1.75-2.25
	Plexiglass	10	6	1.14	0.9-1.3

$N_t$ : Number of trials,  $N_s$ : Number of steps per trial

## 4 Experiments in Self-replication

One of the strengths of a self-reconfiguring modular robot is that it can potentially divide up into smaller robots to perform tasks in parallel. For exploration or surveillance tasks, a large number of small robots may allow for greater efficiency, although this may need to be balanced against greater capability of larger robots.

### 4.1 Division Algorithm

In [3], we presented algorithms with which generic 2D and 3D modular systems can divide up into multiple groups in order to more efficiently explore their environment. Since these algorithms do not rely on a specific actuation model,

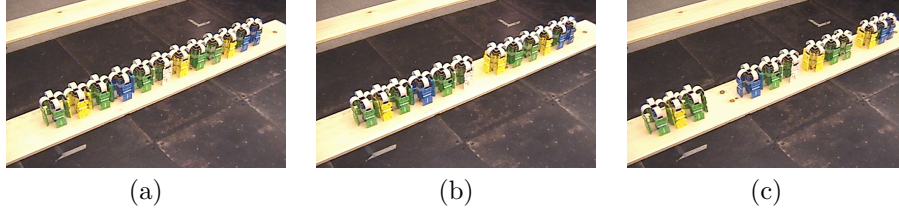


Figure 4: Photographs of recursive division experiment, in which (a) an initial group of 12 modules locomotes, (b) splits into two groups of six modules each, and finally (c) each smaller group splits into two groups of three modules each, with all four groups performing locomotion.

they can be directly applied to the Crystal. In addition, the division process does not rely on synchronization of the modules, so it can be performed in hardware without requiring the addition of any global synchronization.

At a high level, the system can divide due to a command from an outside source or based on a sensor reading in a particular module, after which the two groups can locomote in opposite directions. In each case, a message to that effect is broadcast through the system. For this discussion we assume that the robot is to split itself into a northern group and a southern group. When a module at the north end of the group receives this message, it chooses to be in the northern group, and sets a *signal* variable to 0 to indicate that it is at the edge of the group. A module adjacent to this one will note its neighbor's new direction and also set its *heading* to north, but will set its signal to the value of its neighbor's *signal* incremented by one to note that it is one atom further from the edge of the group. This process continues until the two subgroups come together (not necessarily in the middle). At this point modules may continue changing their *heading* until the signal levels equalize, indicating that the two subgroups are necessarily about the same size. The subgroups then disconnect and move apart.

The use of the signal within the group allows us to show the correctness of this algorithm, as detailed in [3]. Regardless of the relative propagation delays of the *state* messages, no module can initiate actuation until it sees equal signals and opposite headings in itself and its neighbor. This in turn guarantees that the modules divide into two groups of equal size.

## 4.2 Implementation

In order to enable division, we have added more state to the data that is passed between modules, namely the *heading* and *signal*, but no additional message types are required. The current implementation assumes that locomotion will only occur in the north and south directions. An *inch* message with an east or west direction is interpreted as a split message. The general case will require the addition of an explicit *split* message type, which will include the direction



of the division.

As long as the signal level does not exceed 15 (since the current prototype has only 15 modules, this is certain), adding the *heading* and *signal* to the *state* message will not cause it to exceed the two-byte message limit. When a module sets or changes its heading, it immediately sends out its new state to its neighbors to ensure that all modules will receive the necessary data to trigger the division rules. In addition, when a module decides that it is at a valid division point, it will disconnect from its neighbor in the other group only if it has an active connector in that direction. Its neighboring module with the passive connector at the division point will also note that it is at the division point, and simply (and correctly) assume that it will be disconnected.

### 4.3 Experiments

We performed division of groups of six, eight and ten modules in a single column, as well as a rectangle of  $2 \times 6$  modules, both from a static configuration and during locomotion. In all cases (ten trials for each robot size, and regardless of preceding locomotion), the robot divided into two equal-sized groups which then walked apart from each other. In addition, we also performed recursive division experiments in which an initial column of 12 modules was commanded to divide into two groups of six, after which each of those groups was commanded to divide again. This also worked correctly, resulting in four locomoting groups of three modules each. Photos of this experiment are shown in Fig. 4.

## 5 Discussion

These experiments were the first significant repeated tests of untethered unit-compressible modular robots. They were largely successful, in that we were able to successfully perform distributed communication and actuation to achieve specific tasks without any central control using on-board processing and power supply. During the course of the experimentation, we learned several things which we plan to incorporate into future versions of the system. In particular, the connectors are not rigid enough to support some types of reconfigurations, the power consumption could be greatly reduced, and a more general communications protocol would allow for greater algorithmic capability.

Beyond the algorithms presented in this paper, we are also interested in performing reconfigurations with the Crystal. As a first step in this direction, we developed a simple reconfiguration algorithm based on Attaching locomotion, and integrated it with a distributed goal recognition algorithm developed previously. The algorithm performed a single step of locomotion, after which the head of the moving column initiated goal recognition. If the modules collectively determined that they had reached the goal shape, motion stopped, otherwise another step was made. This was seen to work algorithmically, but the current connector design did not allow reliable hardware performance.

## Acknowledgements

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, NSF award IIS-98-18299, and NSF IIS 9912193.

## References

- [1] Z. Butler, S. Byrnes, and D. Rus. Distributed motion planning for modular robots with unit-compressible modules. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, 2001.
- [2] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for a class of self-reconfigurable robots. In *Proc of IEEE ICRA*, 2002.
- [3] Z. Butler, S. Murata, and D. Rus. Distributed replication algorithms for self-reconfiguring modular robots. In *DARS 5*, 2002.
- [4] T. Fukuda and Y. Kawakuchi. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. In *Proc. of IEEE ICRA*, pages 662–7, 1990.
- [5] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Koruda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. of IEEE ICRA*, pages 2858–63, 1998.
- [6] K. Kotay and D. Rus. Locomotion versatility through self-reconfiguration. *Robotics and Autonomous Systems*, 26:217–32, 1999.
- [7] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. Hardware design of modular robotic system. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, pages 2210–7, 2000.
- [8] A. Pamecha, C-J. Chiang, D. Stein, and G. Chirikjian. Design and implementation of metamorphic robots. In *Proc. of the 1996 ASME Design Engineering Technical Conf. and Computers in Engineering Conf.*, 1996.
- [9] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with unit-compressible modules. *Autonomous Robots*, 10(1):107–24, 2001.
- [10] W.-M. Shen, P. Will, and A. Castano. Robot modularity for self-reconfiguration. In *SPIE Conf. on Sensor Fusion and Decentralized Control in Robotic Systems 2*, 1999.
- [11] K. Stoy, W.-M. Shen, and P. Will. Global locomotion from local interaction in self-reconfigurable robots. In *Proc. of IAS-7*, 2002.
- [12] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji. Self-assembly and self-repair method for a distributed mechanical system. *IEEE Trans. on Robotics and Automation*, 15(6):1035–45, Dec. 1999.
- [13] Cem Ünsal and Pradeep Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proc. of IEEE ICRA*, pages 1742–7, 2000.
- [14] S. Vassilvitskii, M. Yim, and J. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Proc. of IEEE ICRA*, 2002.
- [15] M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.