Proceedings of ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2008
August 3-6, 2008, Brooklyn, New York, USA

DETC2008-49535

STOCHASTIC CONTROL FOR SELF-ASSEMBLY OF XBOTS

Nora Ayanian[*], Paul J. White, Ádám Hálász[*], Mark Yim, Vijay Kumar[*]

General Robotics, Automation, Sensing, and Perception (GRASP) Lab
University of Pennsylvania
Philadelphia, Pennsylvania 19104
Email: {nayanian, whitepj, halasz, yim, kumar}@grasp.upenn.edu

ABSTRACT

We propose a stochastic, decentralized algorithm for the self-assembly of a group of modular robots into a geometric shape. The method is inspired by chemical kinetics simulations, particularly, the Gillespie algorithm [1, 2] that is widely used in biochemistry, and is specifically designed for modules with dynamic constraints, such as the XBot [3].

The most important feature of our algorithm is that all modules are identical and all decision making is local. Individual modules decide how to move based only on information available to them and their neighbors and the geometric, kinematic and dynamic constraints. Each module knows the details of the goal configuration, keeps track of its own location, and communicates position information locally with adjacent modules only when modules in their vicinity have reconfigured. We show that this stochastic method leads to trajectories with convergence comparable to those obtained from a brute-force exploration of the state space. However, the computational power (speed and memory) requirements are independent of the number of modules, while the brute-force approach scales quadratically with the number of modules.

We present the schematic of the modules, preliminary experimental results to illustrate the basic moves, and simulation results to demonstrate the efficacy of the algorithm.

[*]We gratefully acknowledge the support from ARO Grant W911NF-05-1-0219 and ONR Grant N00014-07-1-0829. N. Ayanian is supported by a NSF Graduate Research Fellowship. Á. Hálász is supported by the Penn Genome Frontiers Institute. Address all correspondence to N. Ayanian.

INTRODUCTION

The appeal of modular robotics is the potential to build arbitrary objects out of a pile of tiny modules which can reconfigure based on some minimal user input. Smaller modules correlate directly to increased versatility and portability of the system. However, smaller modules increase the number required to form a desired shape. As the number of modules increases, the motion planning search space grows exponentially.

Indeed it is generally very difficult to generate deterministic motion plans for reconfigurations in modular robots except for small numbers of modules. For a large number of modules, the number of configurations grows exponentially [4], and motion planning can quickly become intractable. An alternative to deterministic motion planning is an approach in which all modules are equipped with pre-defined stochastic policies for decentralized reconfiguration decision making. The energy to cause a module to reconfigure can be internal (e.g. servos, motors) or external (e.g. environmental oscillations). In this paper, we discuss a Modular Self-Reconfigurable (MSR) platform which is externally actuated by environmental oscillations, eliminating the typical limitations of modules actuated by internal motors: larger module size, power consumption, complexity, and cost.

Each module uses a set of constraints, knowledge of the goal shape, and the location of modules in its vicinity to determine possible reconfigurations and an associated reward. This stochastic policy is derived from the Gillespie algorithm, which is used to simulate chemical reaction kinetics.

Background

Several groups have demonstrated various algorithms to solve the reconfiguration sequence planning problem ranging from decentralized controllers to global planners, including chain reconfiguration and lattice reconfiguration systems. Murata et al in [5] demonstrated a distributed controller based on the diffusion of state information and module-level global configuration estimation in their Fractum simulation. In [6], Shen et al demonstrate a hormone-based distributed controller as a method for diffusing information to a system of CONRO modules. Yim et al [7] present a "goal-ordering" based distributed controller for Proteo, a general class of 3D Modular Self-Reconfigurable (MSR) modules. In [8], Rus et al present a centralized controller for the Crystalline system. And in [9], Kotay and Rus present a general trajectory planner for the Molecule system and introduce a scaffold planning method that can increase reconfiguration efficiency. Stoy et al [10] present a distributed method where certain seed modules use local communication to establish a recruitment gradient to guide reconfiguration. Pamecha et al [11] implement reconfiguration planner using simulated annealing and a novel metric. In the field of stochastic modular robotics, self-assembly and self-reconfiguration have been controlled by module labeling and task specification [12] and by the use of graph grammars [13].

There are examples in nature of self-assembly of entities with virtually no processing power or sensing capabilities into highly complex aggregates through simple stochastic rules. Examples span many length and time scales, from the synthesis and self-assembly of complicated molecular machines in cells, to sophisticated group behavior of social insects. There has been significant effort to transplant these ideas to engineering, ranging in application from materials, to systems biology, to MEMS, and robotics. The stochastic nature of these phenomena can be emulated using Markov chain Monte Carlo (MCMC) Algorithms [14], which are widespread in engineering. The Gillespie algorithm [1, 2], which emulates the inelastic collisions of individual molecules with the use of Poisson processes, is widely used in chemical applications. Based on the study of the behavior of social insects, a framework was derived for the design of stochastic controllers for swarms of robots, where a version of the Gillespie algorithm is used to describe the swarm [15, 16].

Our method builds on this previous work but differs in two crucial ways. First, we develop a motion planning algorithm for underactuated modules with geometric, kinematic, and dynamic constraints, such as the XBots which we describe in detail below. Second, the proposed algorithm is decentralized, and the computational cost is independent of the number of modules.

The outline of the paper is as follows. We first introduce an example of a MSR platform, the XBot. We then describe the stochastic controller, and present results of simulations, We conclude with a discussion and directions of future work.

THE XBot PLATFORM

The concept of external actuation [3] to drive a MSR robotic system is motivated by a significant challenge in the field of modular robotics: to increase the number and decrease the size of modules. Several groups [7, 17, 18], have demonstrated lattice structured MSR systems at the centimeter scale. Potential applications for smaller modules include higher resolution telepario [19], the 3D representation of objects, and operation in small workspaces.

The extent to which a module's size can be reduced is limited by the mechanisms, power, and processors that a module requires to function. A typical MSR module has two types of actuation mechanisms: a bonding mechanism to make and break bonds with neighboring modules and a reconfiguration mechanism to traverse the robotic structure or relocate the neighboring modules. Generally, reconfiguration mechanism can consume more than half the modules size, weight and power consumption because of the large amount of energy required to reconfigure the module. In addition, they greatly add to a module's cost and complexity.

Using external actuation can relieve the requirement for a bulky reconfiguration mechanism. Rather than enabling modules to reconfigure themselves under their own power, the energy for reconfiguration comes from the system. Without the need for an internal motor or servo for reconfiguration, a module's design and manufacturing can be greatly simplified. In addition, a module's size is no longer constrained by the need for an internal reconfiguration mechanism.

XBot modules bond to one another to form a planar configuration. One module is fixed to the table (shown in Fig. 1) and the other modules form the robotic structure around that fixed module. Each of the four legs of the XBot has a pair of nickel coated neodymium magnets used for bonding and serial communication. Using the magnets, a module bonds to another at the end of two of its legs. To reconfigure, a module uses shape memory alloy wires to actuate a compliant bonding mechanism that breaks the magnetic bond at the end of one leg. This allows the module to pivot 180 degrees about the other magnetic bond. The energy that causes the reconfiguration comes from the oscillations of the table. The table is mounted to an XY stage that can move the table in an arbitrary motion profile. As the table (and the fixed module) cycles through a carefully-defined motion profile, the inertial forces on the reconfiguring module cause it to rotate into its new position. Each motion cycle is sufficient to deterministically cause all types of reconfigurations. In our case, these motions are either clockwise or counterclockwise rotations, involving either a single module or *meta-modules* (a subassembly of modules).

Module reconfiguration is subject to parity, geometric, and connectivity constraints. Each module in an XBot system occupies one of a discrete set of lattice positions. As shown in Fig. 2, the magnetic polarity and serial function (transmit or receive)

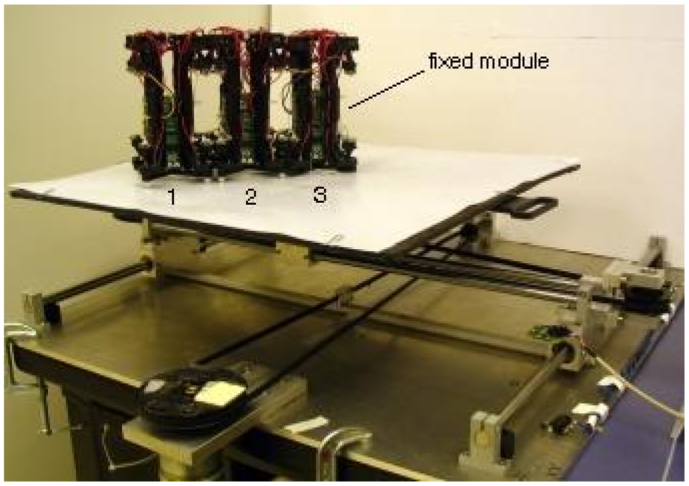

Figure 1: Three XBot modules on the XY stage table. The motion of the table provides the energy for reconfiguration.

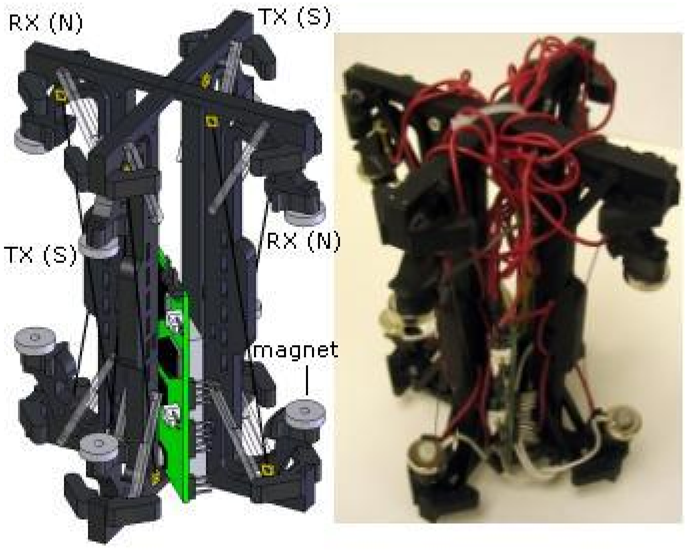

Figure 2: An XBot module. The tops of the module's four legs are labeled indicating the alternation in magnet polarity and serial function. Each leg has either north polarity and receives (RX) or south polarity and transmits (TX). Magnets at the bottom of the leg provide common ground and have polarity opposite to the top magnet.

form an alternating pattern from one leg to the next. Because of this alternating pattern, the grid positions of the configuration space alternate in color, similar to a chess or checker board (see Fig. 3, for two example configurations). When a module reconfigures, it must rotate 180 degrees about its neighbor to a position of the same color as its initial position. Figure 4 shows two examples of reconfiguration: the top panel shows a single module rotating 180 degrees to its next location and the bottom panel shows reconfiguration of a meta-module, a pair of modules working cooperatively, which we will discuss further in the following section. Thus, this parity is maintained throughout the reconfiguration sequence. We discuss in depth the geometric constraints as well as the connectivity constraints in the following section.

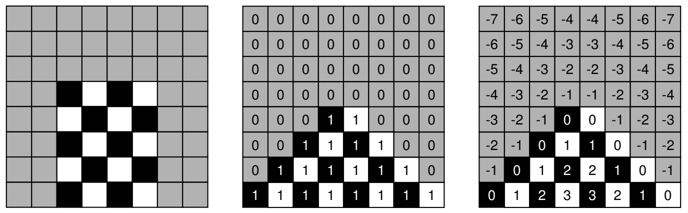

Figure 3: A top view of initial (left) and final (center, right) robot configurations, and two candidate potential maps.

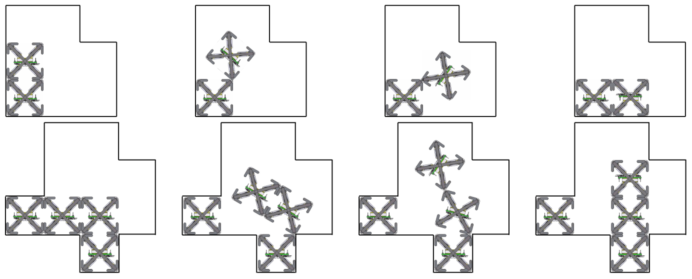

Figure 4: A top view of two module reconfigurations. Top: A single module reconfiguration. Bottom: A meta-module reconfiguration.

METHODS

We consider a population of N modules $M = \{m_k | k = 1,\dots,n\}$ moving in a discrete state space represented by a matrix with entries 0 (empty) or 1 (occupied). Starting from an initial configuration, the modules must reconfigure to a final configuration while maintaining connectivity and obeying all constraints. Figure 3 shows a sample initial configuration (left) and a sample final configuration (center).

Design of the reconfiguration algorithm

Our method can be viewed as a Markov chain Monte Carlo (MCMC) algorithm. The system undergoes a random walk through the space of all possible configurations. As in all MCMC algorithms, the central design issue is the choice of probability distributions from which to sample. The goal is to construct a

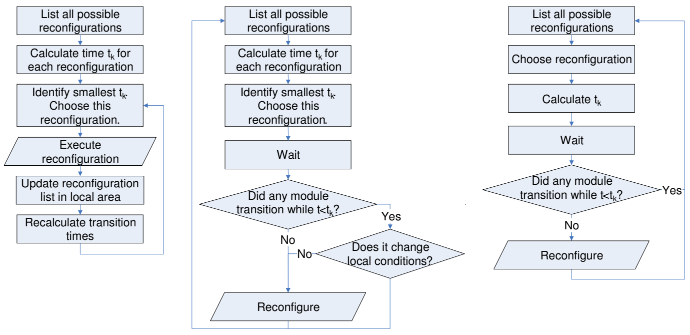
List all possible reconfigurations
Calculate time t_k for each reconfiguration
Identify smallest t_k. Choose this reconfiguration.
Execute reconfiguration
Update reconfiguration list in local area
Recalculate transition times
List all possible reconfigurations
Calculate time t_k for each reconfiguration
Identify smallest t_k. Choose this reconfiguration.
Wait
Did any module transition while $t<t_k$?
Yes
No
Does it change local conditions?
No
Reconfigure
List all possible reconfigurations
Choose reconfiguration
Calculate t_k
Wait
Did any module transition while $t<t_k$?
Yes
No
Reconfigure

Figure 5: Flow charts for the centralized algorithm (left) and two implementations of the decentralized algorithm (center, right). All three are mathematically equivalent.

Markov chain for the system that has the desired configuration as its equilibrium distribution. The probability distribution function is based on a global potential, derived from a local potential map by summation over all modules. The potential map is chosen to maximize global potential at the desired configuration.

To decentralize the MCMC algorithm, we require the distribution to be a Poisson distribution and the process to be a homogeneous Poisson process. Very briefly, we want a stochastic process in which the random reconfiguration events are completely independent, and the average number of reconfiguration events per unit time is a (pre-determined) constant. By requiring each module to have a stochastic decision making process that is a Poisson process, we are guaranteed that the entire system also emulates a Poisson process. This idea is used in the Gillespie algorithm to simulate chemical reactions in well-mixed environments when the number of molecules is small and the continuum models are not applicable.

The Poisson process is characterized by a rate parameter λ and firing times between one to another that are exponentially distributed with the expected time being given by $exp(-\lambda t)$. When there are multiple modules and multiple states, a proposed transition time is generated for each transition from a Poisson distribution using the different rate parameters. The transition with the earliest proposed time is executed; the proposed times for the transitions which are impacted by this reconfiguration are recalculated, and again, the next earliest transition is executed.

The above prescription can be used either as a system-level specification, where all possible transitions of all modules compete against each other, or on the module-level, where modules generate and follow their own proposed transition and time, and recalculate them only if the local configuration changes. The centralized and decentralized implementations, illustrated in Fig. 5 are mathematically equivalent.

The use of competing transition times derived from Poisson processes enables decentralization. In a centralized simulation, the individual generation of Poisson times for each transition can be replaced by the generation of a single transition time. The choice of the transition can be made using any stochastic method which ensures that the likelihood of choosing a particular transition is proportional to its propensity. If, in a simulation, one is interested only in the sequence of steps, the generation of transition times is optional, and the algorithm is a simple MCMC. Similarly, for real-world implementation, the process of generating individual transition times has to be insulated from the physical time it takes to transition. This is a practical issue which, similarly to the question of connectivity, can be addressed by simple, inexpensive means, which would introduce an element of collective communication but would not compromise the advantages of the decentralized method.

The decentralized algorithm

The design of a decentralized algorithm is quite simple. After each reconfiguration which directly affects a module's list of permissible reconfigurations, a module m_k calculates its set of permissible reconfigurations according to the reconfiguration and connectivity rules. The module then calculates the potential difference for each permissible reconfiguration, $i \rightarrow j$, from the potential map, $\Delta_k^{i\rightarrow j} = V(j) - V(i)$ for each reconfiguration, and assigns each a propensity, $p_k^{i\rightarrow j}$, given by

$$p_k^{i\rightarrow j} = \exp(\alpha \Delta_k^{i\rightarrow j}), \tag{1}$$

where α is a scaling factor which modulates the likelihood that potential-lowering reconfigurations will occur. Each module then determines a probability distribution for the reconfigurations on $[0,1]$, generates a uniform random number, and chooses the corresponding reconfiguration.

Next, each module generates a Poisson random number t_k with expected value

$$\tau_k = \frac{1}{\lambda_k} = \left(\sum_j p_k^{i\rightarrow j} \right)^{-1}. \tag{2}$$

and waits t_k time units. If no other module has reconfigured, it communicates its position change to other modules, reconfigures, then repeats the cycle. If another module has reconfigured, it must determine whether the list of possible transitions has changed. If it has not, then it transitions. If it has, then it must update the list, and choose a new time t_k.

Although each module runs an independent Poisson process, the resulting behavior of the entire system obeys a Poisson process due to its properties. Multiple concurrent Poisson processes with intensities $\lambda_1, \ldots\lambda_N$ are equivalent to a single Poisson process with intensity $\sum_{i=1}^{N} \lambda_i$ [20] allowing for our decentralization.

The strong Markov property ensures that if the origin is moved to a random point t_1, then the points of the Poisson process to the right of t_1 form a Poisson process independent of t_1 [20]. This ensures that reconfiguration times need not be recalculated unless conditions for that module's reconfigurations have changed (due to local reconfigurations).

Reconfiguration Rules

Reconfigurations can only occur if there exists a collision-free path to the next configuration; this is determined using geometric rules for the reconfiguration. Each reconfiguration can involve a single module or two modules acting together as a meta-module. The use of meta-modules is an established technique [21, 22] used to simplify the motion planning problem and expand the reachable space of configurations for a MSR system. Without the addition of the two meta-module primitives (defined below), the reachable space for a given XBot configuration is severely confined. Note that the need for a meta-module is suggested by the inherent parity of the system: a module always needs an opposite colored neighbor module to reconfigure about.

In single module reconfigurations, the moving module must ensure that the goal position is empty, and that there exists a collision-free path to that position. In meta-module reconfigurations, one module serves as the leader, and the other a follower. The leader ensures the goal locations are empty and the existence of a collision-free path, and instructs the follower to release the proper inter-module bonds. It is not important which module is the leader, as long as it is a consistent rule for that reconfiguration primitive Both modules in the meta-module do not necessarily reconfigure to a new position. For example, in the meta-module reconfiguration in Fig. 4, the upper right module facilitates the reconfiguration of the center module. This is an example of how the use of meta-modules expands the reachable configuration space and simplifies planning. Without this primitive, the center module would be stuck between the right module and the left module.

Figure 6 shows the geometric constraints for 6 motion primitives considered in our model. The panels, from left to right, show (1) single module, (2) rigid pendulum meta-module, and (3-6) double pendulum meta-module reconfigurations. Modules depicted "m" act in a meta-module to facilitate reconfiguration of the other module. Modules without "m"s which inhabit white boxes can reconfigure to the blue boxes (or vice versa) if the black boxes are occupied and the grey boxes are empty.

Maintaining Connectivity

Each reconfiguration must maintain the connectivity of the system both during and after the reconfiguration. Connectivity is determined on the *connectivity graph*.

The *connectivity graph* on the set of modules is the pair of sets $G = (M, E)$, where $E \subseteq 4[M]$ is the set of edges on the

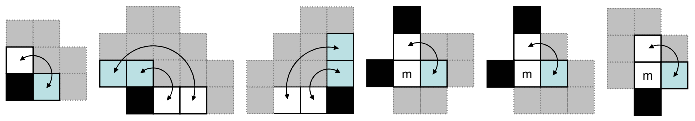

Figure 6: Geometric constraints for 6 possible reconfigurations. Modules depicted "m" act in a meta-module but ultimately do not reposition.

graph. Pairs of modules for which $(m_i, m_j) \in E$ are called adjacent. Each module can be adjacent to a maximum of 4 modules, with which it shares a side.

To ensure the connectivity graph stays connected, the modules m_k which are adjacent to the reconfiguring module m^* must be connected to a fixed module if m^* disconnected. This is typically a centralized calculation, where a leader keeps track of the locations of all modules and determines whether reconfigurations are admissible. In our decentralized approach, each module adjacent to the reconfiguring module(s) sends a signal to the fixed module without using the edges in E which contain module m^*. If that signal can return to the adjacent module, then the reconfiguration maintains connectivity.

To determine which reconfigurations are permissible, it is not necessary for modules to know the state of the entire system, but each module must know the location of other modules in its vicinity. In small systems, modules can keep track of the entire system since it may not be costly. Prior to any reconfigurations modules exchange information about their locations so that each module has a global map. Then, the module m^* sends a signal that it is moving from its current location i to a new location j, along with the current and new location for any follower module. Each module then updates its record of the global state. In large systems, it would be costly for each module to maintain global state information. In this case modules exchange information locally with those in their vicinity, and update the local state information if there are any changes nearby.

Potential Map

The only requirement for the potential map is that the potential must have a unique maximum at the goal configuration. The simplest example of a valid potential map is the center panel in Fig. 3; however, because the potentials for locations outside the goal configuration are equal, there is no reward for moving toward the goal configuration (except on the boundary). A better potential map is the one shown in the right panel of Fig. 3, which rewards moves toward the goal configuration and penalizes moves away from the goal configuration. Although there is no recipe for calculating the potential map, for convex shapes, a heuristic would be to assign zero potential to locations on the outer boundary of the goal shape. Each step into the desired

Table 1: Statistical analysis of 1000 trials with 20 modules

Convergence Point	Mean	Std Dev	Optimal
Two Defects	14.27	7.35	7
Total, no failures	133.1	169.3	17
Total with failures	207.63	486	17

shape would increase potential, and each step out of the shape would decrease the potential (right panel, Fig. 3). One can modify the potential map to speed and/or increase convergence based on observed convergence patterns, which we discuss in more detail in the next section.

RESULTS

In this section we discuss the results of simulations which illustrate the benefits and drawbacks of the method. The simulations run on MATLAB. There are two main issues to focus on. First, we are interested in determining if the desired goal configuration is reached or not. Second, we are interested in convergence rates. It is also worth pointing out that for large numbers of modules, it may be acceptable to reach a configuration that is "close" to the goal configuration. Holes or defects in the assembly may be acceptable. Therefore, we are also interested in convergence rates to configurations that are close to the goal configuration.

Statistical Analysis on Small Groups

To compare our stochastic method with a deterministic method, 1000 simulations were done to statistically determine convergence rates. These simulations were done using the initial configuration in the left panel of Fig. 3 and the final configurations and the potential map of the right panel of Fig. 3. For comparison, we used a breadth-first search (BFS) to determine the shortest path to the goal (shortest means least number of reconfigurations, with meta-module reconfigurations counting as one reconfiguration). The results of this study as well as the BFS are shown in Table 1. The first row describes convergence to two defects, the second row describes total convergence, ignoring the 26 trials which did not converge in under 3000 reconfigurations, and the third row includes those trials which did not converge in 3000 reconfigurations, using 3000 reconfigurations as their total convergence. Before finding the optimal solution, the BFS calculated 165,338 configurations in the state space.

To illustrate the convergence of the system in this study, Fig. 7 shows the convergence pattern. Convergence to two defects is rapid, then convergence slows while modules search for a hole to fill and other modules in goal positions exit and re-enter.

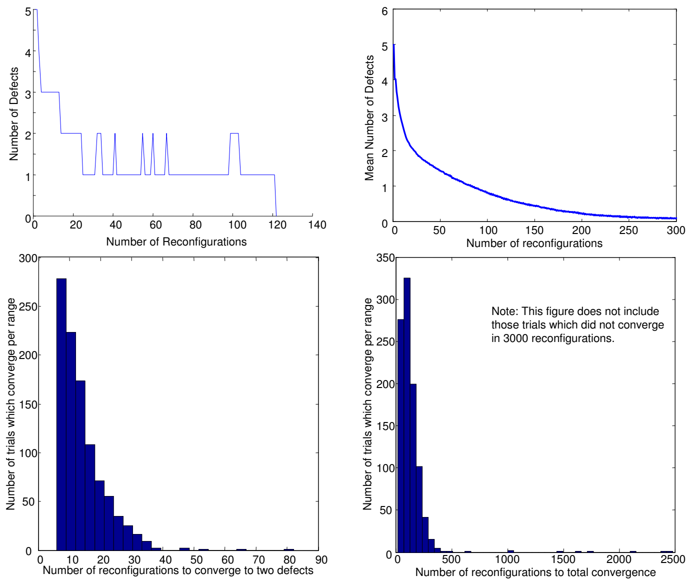

Figure 7: Results from 1000 trials with 20 modules. Top Left: A typical run showing convergence to the goal configuration in 121 steps. Top Right: Averaged results over the 1000 trials showing the mean convergence rate. Bottom: A histogram of time taken to reach configurations with only two defects (left) and the time taken to reach the goal configuration (right).

Larger Groups of Modules

With larger groups of modules, calculating an optimal path to the goal via BFS becomes prohibitively complex. Here our method has a significant advantage over deterministic methods. For the smaller group, the BFS took longer than a day to compute, and the state space scales quadratically with the number of robots. Even in the small case, running the stochastic algorithm takes much less time as well as much less computing power.

Steps Toward Optimizing the Potential Map

Although the heuristic suggested previously may result in total convergence, it may be desirable to modify the potential map to speed the convergence process. Using observations of a few runs as a guide, one can note patterns which end in total convergence and those which do not. With this knowledge, a potential map can be designed to speed total convergence, convergence to a number of defects, or to make pathological cases less frequent.

Pathological Cases. Figure 8 shows pathological cases for groups of 42 and 72 modules. These pathological cases can be made less likely by modifying the potential function based on behaviors observed in a few test runs. For example, to increase the probability that a hole will be filled, either (1) increase

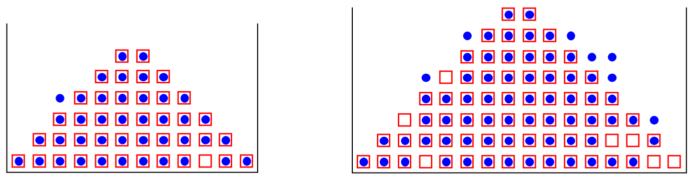

Figure 8: Pathological cases with 42 and 72 modules.

potential value of that location, (2) decrease potential difference between goal locations that block the hole and non-goal locations which would unblock the hole, or (3) increase potential difference for pairs of locations along a typical observed path to that hole location.

It is important to note that in some applications, it may not be critical to have all holes in the interior of the shape be filled. As the number of modules increases and as the size of modules decreases, holes in the interior become less critical.

CONCLUDING REMARKS

We presented a decentralized method for stochastically controlling modular robots to self-assemble into desired shapes. Our algorithm takes into account the geometric, kinematic, and dynamic constraints that are realized in the XBot platform. The algorithm is a MCMC algorithm in which the desired distribution function is based on a global potential which is maximized at the desired configuration. The decision making process at each module is a Poisson process, and minimal communication between modules is necessary.

It is true that a pre-computed deterministic (not necessarily optimal) plan would likely execute in a shorter time with a smaller number of reconfigurations; however, this stochastic approach shows promise as a very low-computation solution. A BFS for a 20-module calculation took longer than a day to compute on a 2 GHz Intel 64-bit processor running Unix and MATLAB, and explored 165,338 configurations before finding an optimal solution. Our decentralized method typically generates results much faster, although the solution can be far from optimal. The key benefit is that the processing power and memory requirements for each module are independent of the number of modules and the desired shape.

One limitation of this algorithm is that total convergence is not guaranteed; however, we have discussed a few ways in which convergence may be increased via the potential map. In the future, we would like to explore the possibility of developing deterministic rules for designing potential maps, eliminating guessing and trial-and-error for non-star-shaped and topologically convex sets. Another limitation of this method is that the last few modules take the longest to converge. One possible way to address this issue is to switch this stochastic controller with another, perhaps more complex, decentralized controller once almost-total convergence was achieved. Even though the controller would be more complex, it would not be as difficult to plan for the handful of wandering modules when almost all modules had reached a goal position. These are directions for ongoing research, as is experimentation with the XBot platform.

REFERENCES

[1] Gillespie, D. T., 1976. “A general method for numerically simulating the stochastic time evolution of coupled chemical reactions”. *Journal of Computational Physics,* **22**(4), pp. 403–434.

[2] Gillespie, D. T., 1977. “Exact stochastic simulation of coupled chemical reactions”. *The Journal of Physical Chemistry,* **81**(25), pp. 2340–2361.

[3] White, P. J., and Yim, M., 2007. “Scalable modular self-reconfigurable robots using external actuation”. In 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2773–2778.

[4] Chen, I.-M., 1994. “Theory and applications of modular reconfigurable robotic systems”. PhD thesis, Division of Engineering and Applied Science, California Institute of Technology,, Pasadena, CA.

[5] Murata, S., Kurokawa, H., and Kokaji, S., 1994. “Self-assembling machine”. In Proceedings of the IEEE International Conference on Robotics and Automation.

[6] Shen, W.-M., Salemi, B., and Will, P., 2000. “Hormones for self-reconfigurable robots”. In Intl. Conf. on Intelligent Autonomous Systems (IAS-6), IOS Press, pp. 918–925.

[7] Yim, M., Zhang, Y., Lamping, J., and Mao, E., 2001. “Distributed control for 3d metamorphosis”. *Autonomous Robots,* **10**(1), p. 41.

[8] Rus, D., and Vona, M., 1999. “Self-reconfiguration planning with compressible unit modules”. In Proceedings of the IEEE Intl. Conference on Robotics and Automation, Vol. 4, pp. 2513–2520.

[9] Kotay, K. D., and Rus, D. L., 2000. “Algorithms for self-reconfiguring molecule motion planning”. In 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vol. vol.3, p. 2184.

[10] Stoy, K., and Nagpal, R., 2004. “Self-reconfiguration using directed growth”. *International Symposium on Distributed Autonomous Robotic Systems*, June 23-25.

[11] Pamecha, A., and Chirikjian, G., 1996. “Useful metric for modular robot motion planning”. In Proceedings of the IEEE International Conference on Robotics and Automation.

[12] White, P., Zykov, V., Bongard, J., and Lipson, H., 2005. “Three dimensional stochastic reconfiguration of modular robots”. In Robotics: Science and Systems, MIT.

[13] Bishop, J., Burden, S., Klavins, E., Kreisberg, R., Malone,

W., Napp, N., and Nguyen, T., 2005. “Self-organizing programmable parts”. In International Conference on Intelligent Robots and Systems, IEEE/RSJ Robotics and Automation Society.

[14] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., 1953. “Equations of state calculations by fast computing machines”. *J. Chem Phys,* **21**, pp. 1087–1092.

[15] Berman, S., Halasz, A., Kumar, V., and Pratt, S., 2006. “Algorithms for analysis and synthesis of a bio-inspired swarm robotic system”. In *Lecture Notes in Computer Science 4433: Swarm Robotics*. Springerlink, pp. 56–70.

[16] Halasz, A., Hsieh, M. A., Berman, S., and Kumar, V., 2007. “Dynamic redistribution of a swarm of robots among multiple sites”. In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems.

[17] Rus, D., and Vona, M., 2001. “Crystalline robots: Self-reconfiguration with compressible unit modules”. *Autonomous Robots,* **10**(1), 01/01, pp. 107–124.

[18] Unsal, C., and Khosla, P. K., 2000. “Mechatronic design of a modular self-reconfiguring robotic system”. In Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 2, pp. 1742–1747.

[19] Goldstein, S. C. ., Campbell, J. D., and Mowry, T. C., 2005. “Programmable matter”. *Computer,* **38**(6), 06, pp. 99–101.

[20] Kingman, J. F. C., 1993. *Poisson Processes*. Oxford University Press, New York.

[21] Vassilvitskii, S., Yim, M., and Suh, J., 2002. “A complete, local and parallel reconfiguration algorithm for cube style modular robots”. In Proceedings of the IEEE International Conference on Robotics and Automation, pp. 117–122.

[22] Christensen, D. J., Ostergaard, E. H., and Lund, H. H., 2004. Metamodule control for the atron self-reconfigurable robotic system.