Distributed locomotion algorithms for self-reconfigurable robots operating on rough terrain

Zack Butler and Daniela Rus
Dept. of Computer Science
Dartmouth College, Hanover, NH, USA

Abstract In this paper, we describe a set of distributed algorithms for self-reconfiguring modular robots that allow them to explore an area in parallel. The algorithms are based on geometric rules that each module evaluates independently relative to its local neighborhood. This paper concentrates on developing algorithms within this framework to enable travel over the widest variety of terrain. In particular, we show how to perform straight-line motion, turning while on obstacles, climbing over tall obstacles, and tunneling under overhangs, all of which work for groups of arbitrary size. This last feature is important, as it also allows a large system of self-reconfiguring modules to divide up into several groups of various sizes, each of which is equally capable of motion and participation in the overall group task. We also discuss implementations and ways to improve efficiency and switching between tasks.

1 Introduction

Self-reconfigurable robots are modular robots that can morph into different shapes without outside assistance. They use this ability to adapt their shape to the task and environment at hand, such as building tall structures to surmount large obstacles and forming into snake-like shapes for navigating tunnels. Self-reconfiguring robots can also function as parallel distributed machines. That is, in almost all existing systems, each module has a processor on-board, and to take best advantage of the modularity, the algorithmic processing as well as module control should take place in a distributed fashion. This also allows the system to be more robust to failures of individual modules and communications, and supports the partition of the robot. Several self-reconfigurable systems have been designed and built, e.g. [4, 7, 8, 12]. Centralized algorithms have been proposed for most [3, 6, 11, 14], but purely distributed algorithms have been less explored [5, 9, 10, 13].

In our work, we are concerned with endowing these systems with the ability to efficiently explore their environment using distributed control. The modules should be able to move over rough terrain, including traversal of tall obstacles as well as small holes, and be able to split up into smaller groups for efficient parallel exploration. An example of simulated task performance is shown in Fig. 1, in which several groups of modules are exploring an open piece of terrain in parallel. In this case, a larger number of groups is desirable to speed up exploration, but small groups cannot navigate large obstacles. The system can use the ability to dynamically change group size and number to more efficiently conduct exploration, while using efficient climbing algorithms to make best use of the available modules in each group.

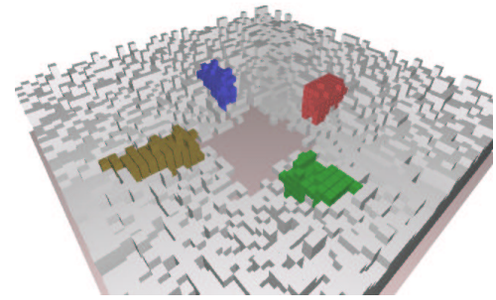

Figure 1: Four groups of modules on rough terrain.

In this paper, we extend previous related work [1, 2] and develop distributed generic algorithms for various facets of locomotion as well as dynamically changing the size and number of robots. By "generic" we mean algorithms that are independent of the specific architecture of the basic robot module. Our algorithms can be instantiated to any self-reconfiguring robot system where an individual module has the ability to move linearly and make convex and concave transitions on a substrate of identical modules. Most existing systems fit this model. A set of geometric rules that is evaluated by each module determines the movement behavior of each cube, and thus the overall behavior of the robot.

1.1 Algorithmic Model

Each algorithm presented here employs a set of local rules we implement as a type of cellular automaton. Each rule requires a set of preconditions on the neighborhood of the cell and when activated, causes the cell to execute a particular action. We represent the basic module of the robot as a cube, but our proposed abstraction can be replaced by other geometric structures that support the formation of lattices. Preconditions may include presence or absence of a module at particular location, or given values of a module's internal state. The result of a rule is either motion of the cell, an update of an internal state, or both.

In traditional cellular automata, the next configuration is generated from the current configuration by simultaneously evaluating all the cells. This model does not consider physical aspects of the underlying system. Since our robots are physical systems, we modify the evaluation model for traditional cellular automata to sequence cell evaluation. In particular, for the algorithms presented here, we use an activation model introduced in [1] and referred to as D_1. Under the D_1 model, a cell can delay activation at most one cycle relative to any other cell, or equivalently, one cell can activate at most

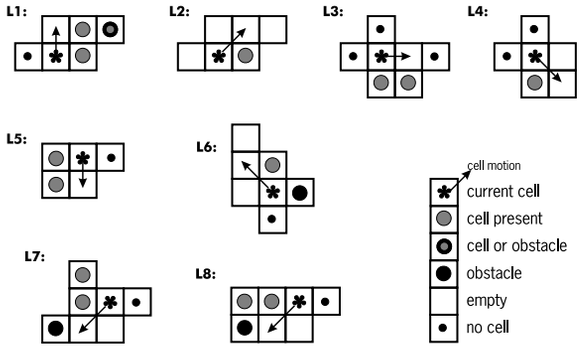

Figure 2: Eight rules for eastward locomotion over obstacles along with legend applying to all rule sets.

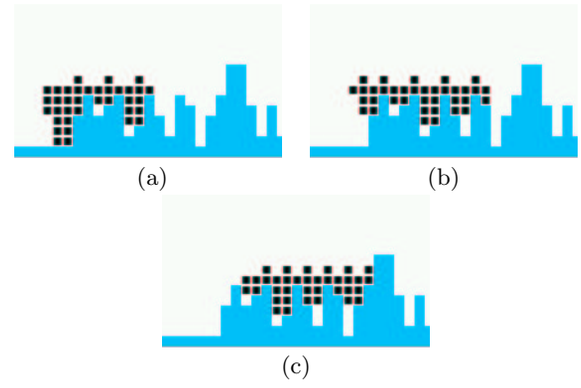

Figure 3: Group using the 8 rule set (a-b) locomotes over obstacles shorter than the initial group height and (c) stops when reaching a taller obstacle.

twice before another cell activates once. This model assumes some homogeneity among the modules of the robot, namely that they run at approximately the same speed. In our most flexible model, D_∞, a cell can delay an arbitrary amount between activations. Rule sets have been developed that are correct under this model, although they tend to be more complex and require more state for a given task. The algorithms discussed in this paper are evaluated under the D_1 model.

2 Locomotion

In previous work [1], a simple set of eight rules was developed that allowed a group of modules to walk in a straight line over low obstacles (shorter than the initial height of the group). This limited the terrains that the robot was capable of traversing. Another rule set presented in [1] allowed climbing of higher obstacles under the less restrictive D_∞ activation model, but used a larger number of rules. Here, we extend the eight-rule locomotion to a set of sixteen rules that enables two new capabilities: (1) climbing over cliffs much taller than the initial group size, and (2) tunneling under overhanging obstacles. This latter extension may be especially important for tasks such as search and rescue operations where the modules will need to navigate small openings.

2.1 Simple locomotion

We first review the basic locomotion algorithm, as it forms the basis for the later extensions. Since this algorithm is simple, it may be useful in its own right for various tasks, such as indoor locomotion. In all the locomotion algorithms, the idea is to produce an overall tank-tread type of locomotion, where modules at the back of the group move over the top to form the next column at front of the group. A series of screen shots from a 2-D simulation (Fig. 3a-b) illustrates this behavior. All the locomotion rules (including the climbing and tunneling extensions presented below) are planar in the plane defined by the direction of motion and the vertical. These rules all extend easily to 3-D groups as explained below, although the snapshots presented are largely from 2-D simulations for clarity.

To generate this type of motion, we use five rules for perfectly flat ground, and eight rules for environments with obstacles. The eight rule set is shown in Fig. 2. We use compass directions to describe the neighborhood, with the assumption that the group as a whole is moving to the east (the rules can be mirrored for movement to the west). Each rule produces motion in a different direction (except for L7 and L8, both of which produce southwest motion). The basic idea is that each module will start moving at the back of the group with north motion, followed by northeast, east, southeast and south motions to reach its new location at the front of the group. It then stays fixed until the group has passed it and it is once again on the west face of the group.

To extend these rules to the 3-D case, it is nearly sufficient simply to run these planar rules in each layer of a 3-D group. However, as a 3-D group travels over uneven terrain, it is possible for one layer to get ahead (or behind) the others, and potentially fragment the group. This is undesirable, and easily avoided. One way to do this is to add additional preconditions in rule L2. The additions prevent a cell from executing this rule if there is a full column of cells behind it in an adjacent layer. This prevents the back end of one layer from being more than one unit ahead of its neighbors.

2.2 Climbing

The locomotion rule set presented above can be proven correct because it maintains a consistent structure over the surface of the group — that is, the top row of the group always consists of a number of cells one or two units apart moving forward. This allows cells to determine their role in the global group from the local shape of the group. However, the top of the group will remain at a constant height, allowing locomotion only over obstacles that are shorter than the group, as seen in Fig. 3. To fully utilize the ability of these modules to climb over obstacles, we must allow the height of the group to change while still allowing the individual cells to understand their current situation within the overall group. Note that simply allowing one cell to climb atop another while moving across the top of the group could lead to undesirable shapes and disconnection of some cells from the group.

We therefore introduce an additional state to each cell which we call the *active* state. This is fairly intuitive — a cell that can recognize itself as being at the back of the group sets its active state and begins motion toward the front of the group. In this way, cells that are moving over the top of the group can recognize both the original locomotion situation (an active cell ahead of it) as well as the situation of Fig. 3c (in which the first cell to reach the tall obstacle should make itself inactive). We

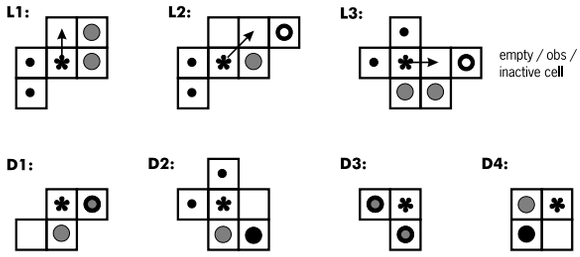

Figure 4: Rules that allow climbing. L1-L3 are replacements for locomotion rules. The other four rules are new rules that are executed only by active cells and result in deactivation of the cell.

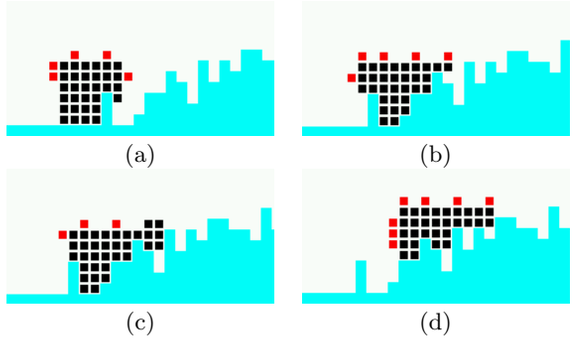

Figure 5: Execution of the climbing rules: (a-b) Before reaching a tall obstacle, the group behaves as under the original locomotion rules. (c-d) When a tall obstacle is reached, the group increases its height just enough to surmount the obstacle.

have developed a set of 13 rules based on this concept which allows climbing over tall obstacles. This rule set is essentially a superset of the eight rule set of Fig. 2, so it is presented as additions and modifications to those rules in Fig. 4. All cells pictured in the rules (other than the current cell) are required to be inactive in order for the rule to be executed. Any locomotion rule will activate the cell if not already active, and only the four rules D1-D4 can deactivate the cell.

The global property of the group under this rule set is that the group height will remain constant as under the eight rule set until a tall obstacle is encountered. At that point, the group will build additional complete rows one at a time until it is one unit taller than the obstacle, as outlined in Fig. 5. Cells can then move over the top of the obstacle and build down the opposite side. The rules presented here then maintain the taller group size after the obstacle, although there are methods (including the use of the tunneling rules described below) to reduce the height of the group for better stability. Like the locomotion rules, the climbing rule set also extends to 3-D groups with the same preconditions on the NE rule.

2.3 Tunneling

For situations such as search and rescue operations, navigation in enclosed spaces is a critical function. For this reason, we have also investigated rules that allow a group to automatically shrink their height in order to tunnel through a small hole. These build on the climbing rules, in that they also use the concept of active and inactive units, and can coexist with the climbing rules in a single rule set. The height of the group generally remains constant, but can be lowered to pass under overhanging obstacles.

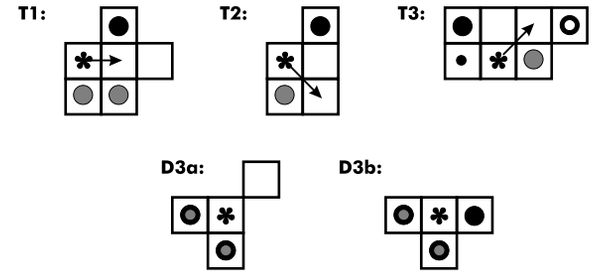

Figure 6: Rules that are added to those of Fig. 4 to allow tunneling. T1-T3 are new rules, while D3a and D3b replace D3 in Fig. 4.

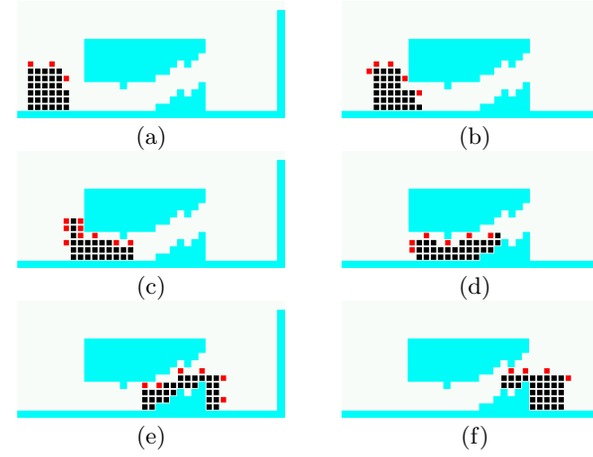

Figure 7: Sequence of snapshots of a simulation in which the group of cells lowers its height to enter a tunnel and climbs to exit the tunnel.

The rule set that enables tunneling is essentially a superset of the climbing rules presented above. Only three additional rules are required, as shown in Fig. 6. The first two specify how an active cell moves under the front of an obstacle to lower the height of the group. These rules allow the behavior seen in Fig. 7a-b. The third new rule enables climbing within a tunnel, as performed in Fig. 7e-f. We also have to make sure that cells do not always deactivate in front of obstacles, since there may be holes under the obstacle, and so we replace one of the deactivation rules (D3 of Fig. 4) with a pair of more specific rules, also shown in Fig. 6. The overall behavior under this rule set is that when a cell is moving down the front face of the group and notices that the wall in front of it has a hole at the bottom, it will not stop in front of the wall, but will move forward and start a new short column under the obstacle.

3 Division and Merging

To take full advantage of the capabilities of self-reconfigurable robots, the need the ability to split up when they need to explore different directions, or when the surveillance task is enhanced by parallelism. The *division* operation splits up a robot into two (in 2-D) or four (in 3-D) smaller groups. This kind of robotic self-replication can

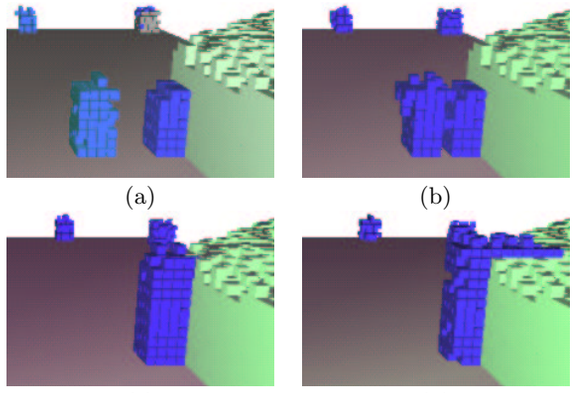

Figure 8: Merging of groups: (a) The group at the right cannot climb the cliff (for this example, we have added constraints to the rules that limit the height the group is able to achieve). (b-c) A second group arrives and merges with the first. (d) The joined group is large enough to climb the cliff even with the added constraints.

be controlled in a distributed fashion, using the same approach as for locomotion. In previous work [2], we presented an algorithm which splits planar groups into two subgroups (which can also be executed on a 3-D system to divide it in half), and one that splits a 3-D group into four subgroups, allowing each subgroup to walk in a different direction.

When a group divides, the resulting subgroups are necessarily smaller. In this paper, we have described a method that allows a group of modules to climb over tall obstacles, but even so, the height of the tallest obstacle that can be overcome is a function of the size of the group. For this reason, division may be counterproductive in very rugged environments. Therefore, we need a way for two smaller groups to easily recombine when needed to climb taller obstacles. We have developed two sets of local rules which allow this behavior.

The set of merging rules presented in [2] applies to two groups meeting head-to-head, and is most suited to having these groups redivide, effectively passing through each other. Our new method of merging groups applies to two groups meeting head-to-tail, that is, with one group catching another. This will happen if a group cannot climb an obstacle and needs to increase in size. This actually does not require any new rules as such, but simply the same climbing rules with a relaxed condition of one previous rule. When a group stops (such as up against a tall obstacle, or for some reason of its own choosing), the modules at the back of the group will be inactive, but will remember their intended direction. Thus, when another subgroup approaches, modules at the front of the group will see inactive units in front of them. We can therefore simply allow the south motion rule (L5 in Fig. 2) to have an inactive cell to the east of the current cell. This is sufficient to allow the merge, at which point the modules in the arriving subgroup will climb over the modules in the first group and they will continue locomotion as one group. This relaxed condition also does not change the behavior of locomotion otherwise, and so this rule can safely be used. An example of this type of merge in action is given in Fig. 8.

4 Turning

In order for a 3-D group of modules to achieve complete locomotion over the plane, it is necessary for the group of modules to change direction. Since the modules are in a square lattice, they will only turn in multiples of 90°, but this is sufficient for full coverage of the plane. We have developed rule sets that enable turning both on flat ground as well as over most obstacles.

The basic set of turning rules, first briefly described in [2], allows turning on any surface that is flat under the footprint of the group. Under these rules, upon recognition that a turn is required, the group will stop and reform into a rectangular prism. The modules will then select the new direction of travel and restart locomotion. Since the group is in a prismatic shape, it is equally capable of moving in any direction from that point, and the turn is easily accomplished.

These rules make use of a new binary state, *halt*, which is used to signal the group that it is time to stop locomotion. This signal is generated by a module moving to the front of the group and is propagated through the system — any module that sees a halted module in front of it will also halt. The halt prevents the start of the back-to-front locomotion process by inhibiting inactive cells from becoming active. Active modules continue their motion, so that a rectangular prism is once again formed. Once a module is halted, it can discern from its neighborhood whether locomotion has completed. Modules on top simply wait for there to be no active modules in their neighborhood, while modules further inside the group wait for the module above them to clear its halt state. When a module turns off its halt state, it also selects a new travel direction. Since the group is prismatic, only the modules at the back can restart locomotion, just as when locomotion first begins, and the group will move in the new direction.

This behavior is implemented with a total of just three new rules. Two of these rules describe how the halt signal is propagated back through the group, and the third detects the departure of moving modules in the neighborhood and chooses the new direction. The new direction is currently hard-coded as a left turn (a hard-coded right turn is also simple), but a higher-level instruction to turn one way or the other could easily be implemented. Propagation of the new direction from the top of the group down is done with four rules that are inherited from the division algorithm but can be used by themselves here as well. In addition, there must be some way to trigger the halting of the group. Currently, this is done through either the presence of a tall cliff in front of the group or an external scripted directive.

4.1 Constant-height locomotion

The previous algorithm can also produce turning on non-flat areas, but will almost always give unsuccessful results. That is, the group will always change direction, but since it will not form a prism first, it will leave some modules behind, or some planes in the new direction may not have sufficient modules for correct locomotion. However, we have developed a second set of locomotion rules which maintains a sheared prism at all times, and therefore allows turning on nearly any surface. As with any increase in capability, this new rule set requires additional state (in this case, knowledge of the initial height of the group), but it is still a purely local algorithm and operates with any group size.

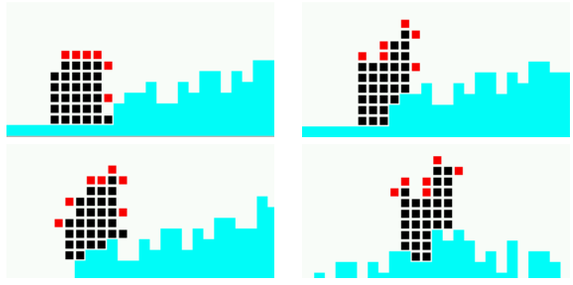

Figure 9: Constant height locomotion. Within the group of inactive (black) cells, each column is the same height (six modules in this example).

The concept behind this rule set is to alter the locomotion rules so that the group will retain its initial height at all points over virtually any obstacle. That is, each stationary column of the group will contain the same number of modules. This property can be seen in 2-D simulation snapshots in Fig. 9. The group will therefore be able to stop locomotion at any point and be in a shape compatible with the turning rules. In order to achieve this, we use a rule set similar to the climbing rules, with the addition that each cell maintains a *height* state along with its *active* state. When a cell reaches the front of the group and deactivates, it will determine its height based on the cell below it, simply by adding one to its neighbor's height. If there is no cell below it, it will set its height to 1 or 2, depending on the local neighborhood.

The rules that result in motion are also generalized from the climbing rules, since the top of the group is no longer flat, but instead matches the underlying terrain. Therefore, an active module may need to move up or down while it moves over the top of the group. We therefore have one set of rules similar to the climbing rules which activate a cell that is at the back of the group. Once a cell is active, however, there are less restrictive rules that force it to move forward over the uneven group. The cell deactivates when it reaches the front of the group, either as in the climbing rules, or when it moves onto a column that is not yet of full height. This requires a total of 18 rules.

To turn under this rule set, the group will behave similarly to the above case on flat ground, requiring some additional rules to accommodate the larger class of possible group shapes. The group will still halt and turn when a module at the front of the group detects a tall cliff or receives a halt instruction. The halt signal is propagated back here as well, although due to the shape of the group, it must be propagated up and down as well as straight back. As before, active modules will continue moving so that the frontmost columns of the group are completed. Whenever a group halts, each column will be the same height as the original group, with certain exceptions. When the group travels over a steep downhill, the modules cannot produce a column of the correct height without looking arbitrarily far down, and so this situation may not produce correct turning. Otherwise, since each column is the same height, and the number of modules is the same as the initial group, there will be no extra modules, and a sheared rectangle will result. The group can therefore be conceptually turned 90° and still have an equal number of modules in each plane.

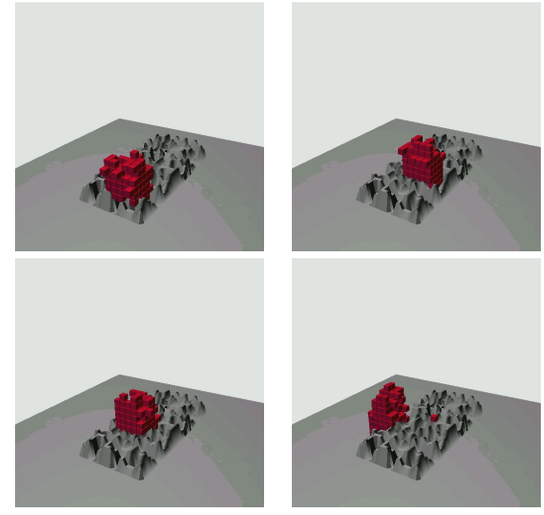

Figure 10: Turning in 3-D over obstacles.

Although the group here can have a more complex shape than the simple rectangular prism used above, the same concepts for turning rules can be used as on flat terrain. However, since the top of the group is no longer flat, the active modules may be in several different positions, and so additional rules are required to detect that there are no active modules in the neighborhood.

Finally, once the modules have selected their new direction, additional state is required to ensure that the group will correctly resume locomotion in the new direction. This is because the modules form a sheared rectangle which is not necessarily a valid starting point for the locomotion rules (although it is a valid intermediate point). An additional boolean state is set by each module indicating that the group has recently turned. A module with this state set has a very restricted set of motions that can guarantee that only modules in the backmost column will begin motion. The state is cleared in all modules through neighbor-to-neighbor propagation once the back column has begun moving, and the regular locomotion can successfully continue.

In order to allow successful turning on the widest variety of terrain, we make one further observation. When a group of modules is moving forward over obstacles, one layer may travel through a narrow crevasse. Consider the case where the group decides to turn while over this crevasse. Since the crevasse is only one module wide, the new planes of locomotion (perpendicular to the previous planes) will contain a single column of modules isolated in the crevasse. These isolated modules cannot come out without disconnecting the group. Therefore, we want to have the modules avoid entering such a crevasse in the first place. We do this by treating any area that is bounded immediately by obstacles on both left and right sides as itself an obstacle. This allows the locomotion to continue correctly and allow correct continuing locomotion after a turn.

5 Implementation / Discussion

Most of the snapshots presented here are from C++ simulations that include both 2-D and 3-D implementations and graphics. We are now developing a new more flexible Java3D implementation, the output of which can be seen in Fig. 10. This framework enables easy change of

rule sets, both before execution and on the fly. The main loop is called by the graphics code, and instantiates the D_1 activation model as described above. Each module is an object that evaluates its current rule set relative to its own situation and moves itself and/or updates its states accordingly.

The rule sets presented above, as well as rules for division, were implemented in a way that allows most of them to coexist as a single large set. This is beneficial in that we can execute several tasks (division, climbing, tunneling, turning) at the appropriate times without having to load different rule sets or handle the transitions between different rule sets. However, when writing such a combined rule set, there is necessarily a prioritization that takes place within it for the various tasks. Also, some actions might be initiated from within the robot (such as turning), rather than as a reaction to the local environment, and this could be more easily handled by changing rule sets for various modules. Since each rule set is an instantiation of a particular Java class, the modules can load in different rule sets in response to different events.

Algorithms based on a collection of rule sets that can be activated or deactivated at different times would enable more flexible behavior of the overall group. In addition, this would make overall execution much more efficient, as each module would only have to evaluate a small number of rules at any given time (the ones pertinent to the active task). The use of collections of rules could still be done in a reactive way by putting overall preconditions on a rule set. For example, a splitting rule set could be used if and only if the module (and its neighbors) have no travel direction. Similarly, if a high wall was encountered, this could signal a transition to turning rule set. Alternately, the use of rule collections would be very useful in "programming" these systems at a high level. To do this, various rule sets would be added in and taken away at the group level to indicate which tasks are to be performed at a specific time. This allows the user to avoid having to interact with each module individually or write specific rules that enable each transition reactively.

6 Conclusion

Self-reconfigurable robots have great potential for parallel exploration and surveillance over very rough terrain. In this work, we have developed several algorithms to enable robots of this type to move over any terrain, crawl through narrow passages, and divide and merge to better accommodate parallel activity. As an example, a system that uses these algorithms would be able to enter a damaged building through a small passageway, and once inside, divide up to quickly search through the rubble. Each group would be able to navigate over the uneven surfaces within. If any group came across a large obstacle such as a wall with a hole near the top, it could call for another group to join it so that the larger merged group could get up to the hole, through the wall and then split up again. This type of system would be much more capable than traditional mobile robots.

In related work, we have instantiated some of these algorithms on to physical robot systems, and hope to continue this work and build systems truly capable of robust exploration. In addition, we plan to build upon these algorithms and develop higher-level algorithms that will control the division and motion of the groups of modules to most efficiently perform various tasks.

Acknowledgments

Robert Fitch developed the underlying engine for the Java3D simulations. Support for this work was provided through NSF awards IRI-9714332, EIA-9901589, IIS-9818299, IIS-9912193 and EIA-0202789 and ONR award N00014-01-1-0675.

References

[1] 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.

[2] Z. Butler, S. Murata, and D. Rus. Distributed replication algorithms for self-reconfiguring modular robots. In *DARS 5*, 2002.

[3] C.-H. Chiang and G. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10(1):91–106, 2001.

[4] T. Fukuda and Y. Kawauchi. 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 IROS*, 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] K. Stoy, W.-M. Shen, and P. Will. Global locomotion from local interaction in self-reconfigurable robots. In *Proc. of IAS-7*, 2002.

[10] 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, 1999.

[11] C. Ünsal, H. Kiliççöte, and P. Khosla. A modular self-reconfigurable bipartite robotic system: Implementation and motion planning. *Autonomous Robots*, 10(1):23–40, 2001.

[12] M. Yim, D. Duff, and K. Roufas. PolyBot: a modular reconfigurable robot. In *Proc. of IEEE ICRA*, 2000.

[13] M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.

[14] E. Yoshida, S. Murata, A. Kaminura, K. Tomita, H. Kurokawa, and S. Kokaji. Motion planning of self-reconfigurable modular robot. In *Proc. of Int'l Symposium on Experimental Robotics*, 2000.