

Distributed Goal Recognition Algorithms for Modular Robots

Zack Butler, Robert Fitch, Daniela Rus and Yuhang Wang

Department of Computer Science

Dartmouth College

{zackb,rfitch,rus,wyh}@cs.dartmouth.edu

Abstract

Modular robots are systems composed of a number of independent units that can be reconfigured to fit the task at hand. When the modules are computationally independent, they form a large distributed system with no central controller. In this paper, we are concerned with the ability of such modular robots to easily recognize the achievement (or lack thereof) of a given goal configuration. We present algorithms for a class of 2D and 3D modular robots, along with correctness and running time analysis. We have successfully implemented the 2D algorithm on the second-generation Crystalline Atomic robot, a self-reconfigurable modular robot under development in our laboratory, and we present implementation details and experimental results.

1 Introduction

Modular robots are composed of discrete components that can be combined, manually or autonomously, in multiple configurations to vary properties such as shape and functionality. Thus, modular robots are more versatile and robust than traditional fixed-architecture robots. In particular, a subclass of modular robots, self-reconfigurable robots, can change shape autonomously to better suit various tasks. Many self-reconfigurable robots comprise homogeneous units [7, 8, 9, 13], allowing them to efficiently form arbitrary geometric shapes without requiring specific plans for each module.

Since modular robots operate as a tightly coupled distributed system, most usage depends on the collective coordination and recognition of the current state of the system. This is especially important in the context of recognizing collectively that the system has achieved its goal. For example, the goal can be a desired shape for a self-reconfiguring application, reaching a certain spot for locomotion or implementing a given configuration for manipulation. These applications are all motivating instances for the distributed goal recognition problem studied in this paper.

Distributed goal recognition is challenging because each module in the system has access to local state information only. This information has to be integrated to form a global picture, and modules must find their positions in this overall state. In

order to compute the solution, the union of these configurations has to be compared against the goal. Our main motivation is to enable a distributed approach to shape generation for self-reconfigurable robots. Unlike centralized solutions, in which algorithms have global knowledge of the positions of all modules, decentralized algorithms for modular robots assume only local knowledge and local communication ability for the modules.

In this paper we develop a solution to the distributed goal recognition problem. We present our algorithms for both 2D and 3D lattice-based systems, prove their correctness, and discuss their implementation in simulation. Unlike many existing decentralized planning algorithms, which suffer from a lack of convergence guarantees or limitations on the classes of goal shapes attainable, our algorithms allow a system to explicitly recognize any goal configuration efficiently. We also describe the implementation of our algorithm in a physical system, the Crystalline atomic robot (Crystal) [9]. The Crystal is a unit-compressible self-reconfigurable robot with homogeneous units. We describe our communication and goal recognition experiments, which demonstrate empirically that our solution is efficient, robust and easy to implement.

2 Related Work

Modular robots have been designed and constructed by a growing number of groups. Many of these are lattice-based systems [2, 6, 7, 12], but other architectures have also been investigated [10, 13]. The Crystal robot used in this paper was developed in our lab, and the concept and original hardware are described in [9]. Configuration planning work based on centralized control has been investigated for several systems [6, 8, 15]. Distributed algorithms for construction of symmetrical shapes were developed by Tomita et al. [11], and distributed algorithms by Hosokawa et al. [5] can achieve a small number of different shapes, but without explicit detection of the configuration. The PacMan algorithm of Butler et al. [1] can perform distributed reconfiguration but requires manual alignment of the goal shape to the robot. The possibility of using goal recognition as a frequently called subroutine is mentioned in [14].

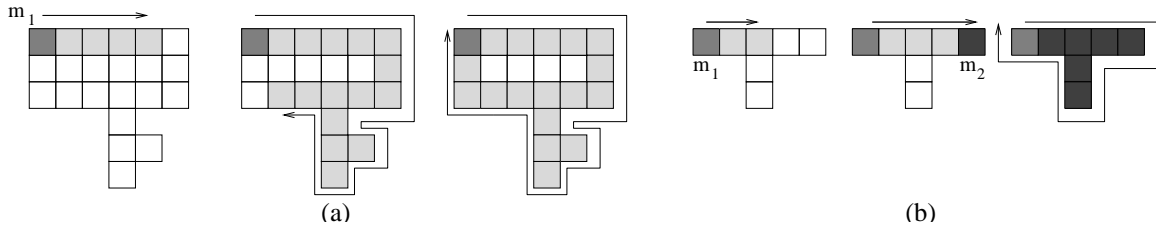


Figure 1: 2D goal recognition examples. In (a), module m_1 sends a successful trace message. Modules that have received the trace are shown in light gray. Arrows indicate direction of the trace. The goal in part (b) has no module at m_2 , causing the trace to fail. Dark modules indicate TRACE_FAIL propagation.

3 Algorithms and Analysis

We now present our approach for solving the goal recognition problem. We outline the main idea of the solution, develop the algorithm for 2D robots, then propose a 3D version. Both algorithms use the same resource upper bounds, $O(n^2)$ total messages executing in (parallel) $O(n)$ time. Finally we describe the implementation of our solution in simulation.

The general goal recognition problem asks whether a modular robot’s configuration matches a particular goal shape. Systems under consideration are constructed from homogeneous modules and their configurations can thus be represented as a binary matrix, with 0 corresponding to empty space and 1 corresponding to space occupied by a module. This representation leads to the following problem formulation: given an oriented goal matrix G and a modular robot A , determine if A ’s configuration matches that specified by G . We assume that the robot is a purely distributed system comprising modules that have local communication, limited processing power and memory, and that form a lattice. To simplify presentation, we consider square (in 2D) and cubic (in 3D) module shapes, but the algorithm works with other shapes as well.

Our solution to distributed goal recognition is based on a technique we call a *trace*. Intuitively, a trace is a tour of the modules of the robot matched at each step against the goal matrix. Consider the situation where one module is assigned a position in the goal matrix. If the matrix has a 1 in that position, then the module is considered *valid*. That module then passes a message to a neighbor, including an indication of the matrix position corresponding to that neighbor. This new module then decides whether it is valid, and so on. If any modules are not valid, the trace is said to fail. Conversely, if all modules are valid, then the trace is said to succeed, and under certain conditions this implies that the robot is positively in the goal configuration. In particular, if the number of modules in the robot and in the goal matrix are the same, and both are fully connected, then a successful trace is both necessary and sufficient to solve the goal recognition problem. Of course, the problem remains as to how any module knows whether all other modules are valid. We approach this differently in 2D and 3D, but in both cases the message passing policy guarantees that the module originating the trace eventually receives the “answer.”

The main idea of our algorithm is for multiple modules to initiate traces in parallel, each testing themselves against the same well-known position in the matrix. We call this position the *anchor*, and arbitrarily define it as the upper-left corner (in 3D, forward upper-left) of the target shape. The modules find this position by scanning the matrix for the first 1. In 2D, a module matching this local configuration would have no north or west neighbors, so any such physical module is called *special* and knows to initiate a trace when the algorithm begins. At most one trace will succeed, and the winning module then sends a global message. If all traces fail, however, then the situation is slightly more complex since no single module has enough information to discern global failure. If at least one module knows that all traces have failed, it can then propagate a global failure message. Therefore, when a trace fails, its originator sends a failure message that acts as a “meta” trace. Any special modules that receive this message hold it until their respective traces fail, then send as normal. When the module gets the meta-trace back, it knows that all other traces have failed and it can propagate the global failure message.

Since the algorithm is distributed, the method of signaling the algorithm’s result is not immediately clear. If the algorithm is used as a subroutine in a larger context, then the first module to attain the global answer could return the result. In this case, however we are working with goal recognition in a stand-alone fashion so we choose to simply propagate the answer throughout the system and program the modules to execute a predefined behavior to signal success or failure.

This algorithm has several nice properties. First, it is distributed and avoids the concept of a supermodule. It also requires less space than simpler solutions. For example, one such simple approach would be that each module broadcasts (through local message passing) its ID and the IDs of its neighbors. When one module has received a message from all modules, it builds a connectivity graph, converts it into a binary matrix, and compares against the goal matrix to compute the answer. That solution requires linear space in each module, or $O(n^2)$ space overall versus our algorithm which requires only linear space overall. Our solution also requires fewer messages. We now describe the algorithm in detail.

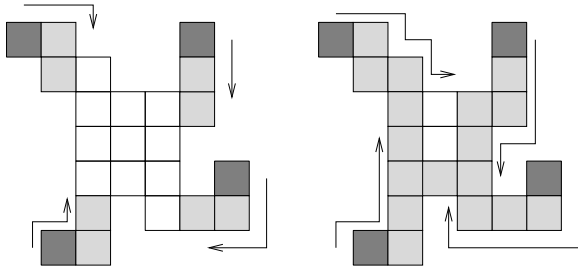


Figure 2: Multiple special modules. Dark modules are special and initiate traces in parallel, with paths of the traces indicated by arrows.

3.1 Goal Recognition for Planar Robots

In two dimensions, note that if the target shape has no “holes,” and the number of modules in the robot and target are equal, then a trace needs only to compare the *perimeter* modules against the goal. This is easily accomplished by passing the trace messages according to the *right-hand rule*. We use the reverse order for convenience so a more accurate term would be *left-hand rule*. Passing messages along the perimeter also insures that the trace message always returns to the originator after being seen by all perimeter modules. Modules pass messages around the perimeter using a `LeftHandPass` function that takes the direction of the incoming message and sends to the next module in a clockwise direction (potentially the previous sender).

The algorithm begins when a message (`GR_START`) is generated from an outside source, such as an outer algorithm using Goal Recognition as a subroutine, and is propagated through the system. Special modules initiate traces in response to the start message, and a successful trace prompts a success message. Failed traces otherwise generated meta-traces, eventually resulting in a global failure message. Behavior to signal success or failure could be implemented as a return to the calling algorithm. Pseudocode for the algorithm, listed as a series of message handlers, is given in Algorithm 1.

Examples with only one special module are given in Figure 1. Part (a) shows a simple shape. Once `GR_START` is received, it propagates through the robot. Meanwhile, m_1 initiates a `TRACE` message, which is valid for each module. When the `TRACE` returns to m_1 , `GR_SUCCESS` is sent and the robot signals success. In (b), the trace fails at m_2 and `TRACE_FAIL` is sent. Module m_1 receives the `TRACE_FAIL`, sends the `META_TRACE`, and then follows with `GR_FAILURE`. Figure 2 is a more complex example with multiple special modules.

3.1.1 Analysis: Our algorithm correctly solves the goal recognition problem for all instances in which the robot and goal shape have the same number of modules and no holes. This can be shown by considering two cases. First, if the robot is in the goal configuration, the anchor position in the matrix must match one special module. This module will initiate a trace that necessarily returns and is valid, at

Algorithm 1 2D Goal Recognition message handlers.

```

GR_START:
  PostMessage(GR_START)
  if I am special then
    LeftHandPass(TRACE, receivedFrom)

TRACE(vector goalPos):
  if message is from me then
    PostMessage(GR_SUCCESS)
  else
    if valid(goalPos) then
      LeftHandPass(TRACE(neighborPosition, received-
        From))
    else
      LeftHandPass(TRACE_FAIL, receivedFrom)

TRACE_FAIL:
  if message is from me then
    LeftHandPass(META_TRACE, receivedFrom)
  else
    LeftHandPass(TRACE_FAIL, receivedFrom)

META_TRACE:
  if message is from me then
    Signal(FALSE)
    PostMessage(GR_FAILURE)
  else
    if I am special then
      while my trace has not returned do
        wait for my trace message
      LeftHandPass(META_TRACE, receivedFrom)

```

which point global success is signaled. If the robot does not match the goal shape, the perimeter of the goal cannot match the robot’s perimeter. Therefore, all traces must fail, but regardless of the robot shape will return to their parent modules. A `META_TRACE` message is then allowed to pass through all special modules to its originator, and global failure is signaled.

The total number of messages is $O(sk)$ trace messages (s traces of k messages each), where s is the number of special modules and k is the number of modules on the perimeter, plus $O(n)$ messages for propagating the solution, or $O(n + sk)$ total. Since $s < k \leq n$, the overall upper bound on number of messages is $O(n^2)$. Since messages are sent in parallel, the time requirement is linear in the number of modules. The space requirement is constant per module, or linear overall.

3.1.2 Extension to General Shapes: The 2D algorithm we presented works under the assumption that the structure has no holes. However, with a simple extension we can be sensitive to holes as well. The addition is another layer of validation messages prior to the global success signal. If a trace is successful (note that even with holes present, only one trace can succeed), its originator propagates a message

through the modules similar to the 3D trace described below. If any interior modules are invalid at this time, they can immediately broadcast GR_FAILURE, while if the message returns to the originator, it can signal GR_SUCCESS. This increases the number of messages by a small factor but does not affect the asymptotic analysis.

3.2 Goal Recognition for 3D Robots

The approach in Algorithm 1 can be used with a 3D robot. The major difference is in how a trace message is propagated. It is unclear how to propagate the trace over the *surface* of a 3D robot and guarantee that the special module receives the correct answer, so we must send the trace to every module instead, using a breadth-first search (BFS) scheme. We must still ensure that the special module retrieves the answer to the trace. This can be implemented with BFS on a spanning tree of the module connectivity graph. First, the special module sends the trace to all its neighbors, and waits for an answer from each. When it receives the trace back from each neighbor, its global answer is GR_SUCCESS if its trace is successful, and otherwise it broadcasts a TRACE_FAIL message. When a neighbor module receives a trace message, it checks to see if it has already received this message. If so, it simply returns TRACE_REPLY(FALSE). Else, it recursively sends the trace to all its neighbors, and so on. The algorithm is listed in pseudocode as Algorithm 2.

In order to detect global failure, during the propagation of trace messages each special module will increment a counter associated with the message. This way, when a special module gets its trace back, it knows how many special modules are in the current configuration. Then, if a special module receives as many TRACE_FAIL messages as traces, it sends GR_FAILURE. If a trace succeeds, the module sends GR_SUCCESS. In either case, the algorithm terminates. Note that since we send messages to all modules, this version is sensitive to holes in the structure.

The 3D algorithm is correct for all robot and goal shapes with equal numbers of modules. As in the 2D case, if the robot matches the goal configuration, exactly one trace will succeed. Otherwise all traces fail, and each special module knows the total number of special modules. Each special module therefore waits until all traces are known to have failed, and then correctly signals global failure.

The number of messages is now $O(n^2)$, and time is still linear. Space requirements are increased, however, to $O(n)$ per module.

3.3 Simulation

We implemented both the 2D and 3D versions of our algorithm using the Java programming language. Snapshots from the 2D version are shown in Figure 3. The user interface provides a simple facility to create robot shapes and target shapes, and animates the messages being passed by changing the color of the modules. The 3D simulator was implemented using the Java3D API. Screen shots are shown in Figure 4.

Algorithm 2 3D Goal Recognition.

```

GR_START:
PostMessage(GR_START)
if I am special then
    for all neighbor modules  $m$  do
        PostMessage(TRACE)

TRACE(vector goalPos):
if already received then
    SendMessage(receivedFrom, TRACE_REPLY(FALSE))
else
    for all neighbor modules  $m$  do
        SendMessage( $m$ , TRACE(neighborPosition))

TRACE_REPLY(Bool bSucceeded, int numSpecial)
if message is from me then
    if bSucceeded then
        PostMessage(GR_SUCCESS)
    else
        numTraces = numSpecial
        PostMessage(TRACE_FAIL)
else
    if received reply from all neighbors then
        if I am special then
            numSpecial = numSpecial + 1
            SendMessage(parent, TRACE_REPLY(answer, numSpecial))

TRACE_FAIL:
numTraceFails = numTraceFails+1
if numTraceFails == numTraces then
    Signal(FALSE)
    PostMessage(GR_FAILURE)

```

4 Experiments

We implemented the 2D Goal Recognition algorithm in hardware using the new prototype of the Crystalline Atomic modules. The first version of the Crystal [9] was constructed in our lab as a physical realization of a unit-compressible self-reconfigurable system. The square units attach to each other at each of four faces, and expand/contract in two dimensions. These properties allow the system to approximate arbitrary 2D shapes. The new Crystal robot includes inter-module communication, sensor inputs for analog and digital sensors, and independent contraction in x - and y - dimensions. Previously we have developed algorithms for Crystals, including reconfiguration planning [1] and self-repair [4, 3]. The new communication capabilities make it possible to implement more complex distributed applications and require the ability to do distributed goal recognition. We have implemented Algorithm 1 on this new hardware. To enable the algorithm, we developed message-handling routines as described below, and performed experimental studies with several different configurations of the modules.

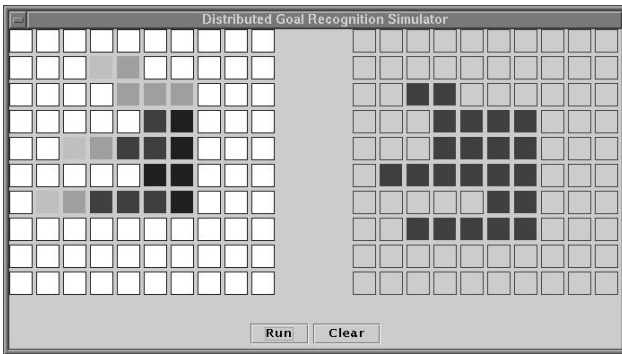


Figure 3: Screenshot from 2D simulator. Left pane represents robot, right pane is goal shape. Special modules are lightened. This instance will fail.

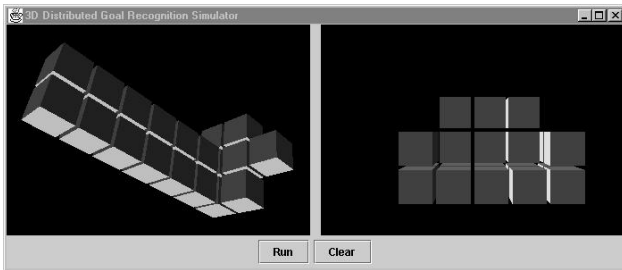


Figure 4: 3D simulator screenshot. The algorithm compares the robot on the left to the goal shape on the right.

4.1 Crystalline Module Hardware

The main functional improvements in the new Crystal module (see Figure 5) are a faster processor, inter-module IR communication capability and sensor inputs. A central control beacon is no longer present to coordinate behavior. Thus, the robot control is now done in a purely distributed fashion. The North/South and East/West faces are now independently actuated, so the degrees of freedom increase to four (two for expansion/contraction, and two for connectors on active faces). A labelled photograph of one module is given in Figure 5.

Each atom's on-board electronics provide computation, IR communication, sensor inputs and motor control for the Crystal's four degrees of freedom. The processor is a Hitachi HD64F3644H running at 10 MHz. Communication is implemented via IR components on the Crystal faces, so a unit can talk with any neighboring units. Each module face contains an IR emitter and detector, connected to a dedicated Maxim Max3100 UART in the core. The UART communicates at 1200 Baud and has an eight-word FIFO. Each face is connected to the core circuit board with a ribbon connector. Connectors on the module core provide four inputs for analog or digital sensors. Four 3V Lithium batteries power the unit and enable fully untethered operation. Code is downloaded to the processor through a serial interface, then executes as soon as the unit is powered on.

The expansion/contraction mechanism uses a rack-and-pinion to actuate each pair of module faces. Two MicroMo motors

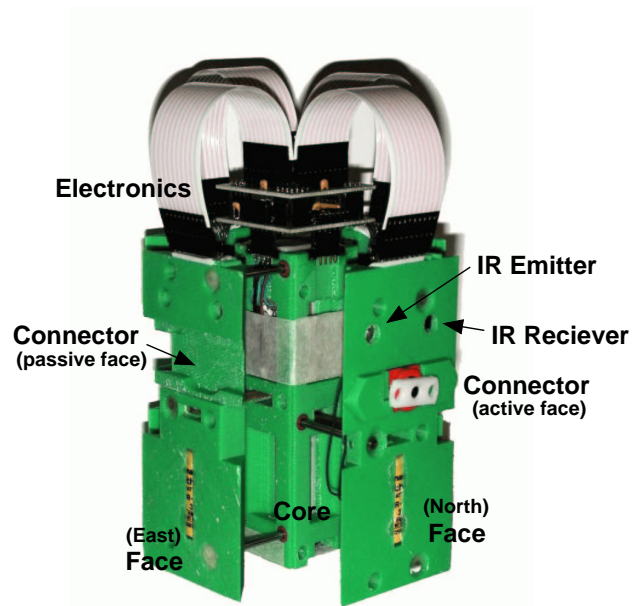


Figure 5: Crystalline Atomic Module, second prototype.

are mounted coaxially in the module core, with pinion gears mounted on the motor output shafts. Racks connected to opposing faces mate on opposite sides of a pinion, so each motor drives two faces simultaneously. Shaft encoders built into the MicroMo's housing generate interrupts which allow the processor to detect when the face is fully expanded or contracted.

Modules attach to each other at their faces, using channel-and-key type connectors. Each module has two faces with active connectors, and two faces with passive connectors. Passive faces simply contain a channel that accepts a bar from an active face. The active face can rotate the bar a quarter-turn, locking the the two modules together, and unlock the modules by reversing the rotation. Lego mini-motors are used to actuate the active faces.

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.

4.2 Algorithm Implementation

In this section we briefly describe the software infrastructure developed to enable the goal recognition algorithms. Since a C compiler is available for the microcontroller we chose, programming the Crystal is done on a host computer in C, compiled, and downloaded to the unit with a serial interface. The robot is a homogeneous system, so each unit runs the same code.

In order to utilize the Crystal's communication capabilities, we created a message passing infrastructure. Each unit has a message queue, and can post messages to neighbor modules.

| sender | data | | | | parent ID | type | | | |
|--------|------|---|---|---|-----------|------|---|---|---|
| 15 | 13 | | | | 7 | 3 | | | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| | | | | | | | | | |

Figure 6: Message format and example. The sample message indicates a trace message received from the west face originating from module 6 with matrix position 51.

We implemented a message loop, similar to the message loop in modern windowing systems. In each iteration, the processor polls each UART for incoming messages and adds any new messages to the queue. It then takes a message from the queue and processes it according to the appropriate message handler. Each UART has its own FIFO, so while the processor is busy handling messages the UARTs still can receive data. Since the processor speed is much faster than the UART transmission rate, the risk of the UART FIFOs filling up before they get serviced by the processor has not been an issue.

The modules must be manually switched on one at a time. When a module comes up, it does not initially know whether its neighbor units are powered on yet. To solve with this problem, we implemented a software “boot” sequence using a special message called SYSTEM_INIT. This message is initially generated only after all modules have been powered on, and can be sent at later times to effect a software “reboot.” This message is created by one module that has a switch. The SYSTEM_INIT message handler performs any initialization necessary, including checking for neighbors using a ping message, and propagates the message to all neighbors. If further SYSTEM_INITs are received, they are ignored. However, to enable later soft reboots, we added a PRE_INIT message to set a state bit that is cleared by SYSTEM_INIT. That way we know when a SYSTEM_INIT (or PRE_INIT) is a duplicate and when it indicates a new reboot of the system. The message format is listed in Figure 6.

Using this infrastructure we implemented the algorithm by creating message handlers for the various required message types, summarized in Table 1. We hardcoded the goal matrix for now, since the message size we chose is so small. Eventually we will transmit goal matrices dynamically, but that requires a more complex protocol to allow for arbitrarily large messages. When a start message is received, the goal recognition algorithm is started and trace messages cause the module LEDs to blink. When the algorithm finishes, the modules signal success or failure with a given pattern of LED blinks. A sample robot is shown in Figure 7.

4.3 Results

We executed the 2D Goal Recognition algorithm using various configurations of the Crystal robot. Data is given in Table 2. Most trials were successful, but since the communication uses IR, sometimes module misalignment led to communications failure. Fortunately when this occurred the trace message

| Name | Function |
|-------------|----------------------------------|
| PRE_INIT | Prepare for initialization |
| SYSTEM_INIT | Perform system initialization |
| GR_START | Begin goal recognition algorithm |
| TRACE | Tests validity in goal matrix |
| TRACE_FAIL | Trace has failed |
| META_TRACE | Check if all traces failed |
| GR_SUCCESS | Return true |
| GR_FAILURE | Return false |

Table 1: List of all message types.



Figure 7: Experimental setup corresponding to rightmost column in Table 2.

continued as if the neighbor module did not exist, and the algorithm correctly recognized the subset of the goal shape.

The next experiment we conducted investigated the behavior of our Goal Recognition algorithm as a subroutine of a simple reconfiguration algorithm. One column of modules was programmed to “inchworm” along a fixed group of modules and automatically initiate Goal Recognition at the end of each inchworm step. Reconfiguration halts when the robot recognizes achievement of the goal shape, in this case a rectangle. Preliminary results indicate the algorithm runs much faster than actuation, although we would like to use Goal Recognition less often during reconfiguration.

5 Discussion and Future Work

In this paper, we have presented a simple algorithm for goal recognition in modular robotic systems. Our solution allows such a robot to compare its overall physical shape with a given goal configuration represented as a binary matrix. We implemented the algorithm in simulation in both 2D and 3D versions, and also conducted hardware experiments using a self-reconfigurable system developed in our lab.






| |  |  |  |  |  |
|----------------------|---|---|---|---|---|
| Matches Goal | No | Yes | Yes | Yes | No |
| Trials | 50 | 50 | 50 | 50 | 50 |
| Successes | 49 | 49 | 50 | 50 | 48 |
| Message Count | 25 | 16 | 67 | 32 | 44 |

Table 2: 2D Goal Recognition for various robot hardware configurations. Column headers represent tested robot shape. “Matches Goal” indicates whether the robot configuration matched the goal for that experiment. Message counts aggregate messages from all modules during simulations of the same module configurations.

An interesting extension would be, instead of computing a binary result, to measure how close the robot is to the goal, using some metric that could be computed in a cumulative fashion by the trace message. For example, the trace could carry a counter of valid versus invalid modules, and the special modules could then use that information in some way, perhaps as input to an outer reconfiguration planning algorithm. In the future we would like to explore this idea in the context of algorithms such as PacMan [1]. Hopefully this simple solution will provide for more effective reconfiguration planning in modular robots.

Acknowledgements

This work was done in the Dartmouth Robotics Lab. Support for this work was provided through the NSF CAREER award IRI-9624286 and NSF awards IRI-9714332, EIA-9901589, IIS-9818299, and IIS-9912193, and a NASA Spacegrant award. We are very grateful.

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] C.-H. Chiang and G. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10(1):91–106, 2001.
- [3] R. Fitch, Z. Butler, and D. Rus. 3D rectilinear motion planning with minimum bend paths. In *Proc. of the Int’l Conf. on Intelligent Robots and Systems*, 2001.
- [4] R. Fitch, D. Rus, and M. Vona. A basis for self-repair robots using self-reconfiguring crystal modules. In *Intelligent Autonomous Systems 6*, 2000.
- [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, I. Ebert-Uphoff, and G. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Trans. on Robotics and Automation*, 13(4):531–45, 1997.
- [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. 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.
- [12] Cem Ünsal and Pradeep Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proc. of IEEE Int’l Conf. on Robotics and Automation*, pages 1742–7, 2000.
- [13] M. Yim. A reconfigurable modular robot with multiple modes of locomotion. In *Proc. of JSME Conf. on Advanced Mechatronics*, Tokyo, 1993.
- [14] M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.
- [15] 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.