Distributed Control for 3D Metamorphosis

MARK YIM, YING ZHANG, JOHN LAMPING, ERIC MAO*
Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304

{yim, yzhang}@parc.xerox.com
lamping@acm.org
ericmao@robotics.stanford.edu

Abstract. In this paper, we define **Proteo** as a class of three-dimensional (3D) metamorphic robotic system capable of approximating arbitrary 3D shapes by utilizing repeated modules. Each **Proteo** module contains embedded sensors, actuators and a controller, and each resides in a 3D grid space. A module can move itself to one of its open neighbor sites under certain motion constraints. Distributed control for the self-reconfiguration of such robots is an interesting and challenging problem. We present a class of distributed control algorithms for the reconfiguration of **Proteo** robots based on the "goal-ordering" mechanism. Performance results are shown for experiments of these algorithms in a simulation environment, and the properties of these algorithms are analyzed.

Keywords: metamorphic robots, distributed control of shape reconfiguration, motion constraints, goal ordering

1. Introduction and Related Work

Metamorphic robotic systems were originally proposed and studied in (Chirikjian, 1993) and (Murata et. al, 1994), where two types of two-dimensional (2D) metamorphic robotic systems were developed. A metamorphic robotic system (Chirikjian, 1993) is a collection of independently controlled mechatronic modules capable of approximating *arbitrary* shape, each of which has the ability to connect, disconnect and climb over adjacent modules. Metamorphic robotic systems share the following properties:

- [**Uniformity**] All modules have the same structure mechanically and electronically, typically the physical structure of each module has some symmetry as well.
- [**Connectivity**] Modules are connected at all times, so the collection of modules acts as a single robotic system.
- [**Mobility**] Each module has the ability to maneuver, so that the system is *self*-reconfigurable.
- [**Locality**] Each module is embedded with a local processor; communication occurs between adjacent modules.

This type of homogeneous system is highly advocated (Murata et. al, 1994) for its robustness, adaptability and mass production. Possible applications range from flexible manipulation to emergent structures and electronic sculptures.

In addition to these metamorphic systems, other work in modular reconfigurable robots include (Fukuda and Kawauchi, 1990), (Paredis and Khosla, 1993), (Yim, 1994), (Hamlin and Sanderson, 1996), (Will et al, 1999) and (Unsal and Khosla, 2000). The earlier of these works provided the impetus and physical proof of concept for modular reconfigurable systems. Self-reconfigurability that comes from metamorphic systems extends the autonomous versatility of these systems.

1.1 Overview

The problem of self-reconfiguration is to rearrange the modules to change from one overall shape to another, without external assistance. This paper studies methods for distributed self-reconfiguration control of three-dimensional (3D) metamorphic robotic systems.

In order to make the methods general, a generic model for a class of metamorphic robotic systems, called **Proteo**, is defined. A **Proteo** model includes the configuration space in which the modules reside and constraints on how modules can move. The model is

* author's current address: Department of Computer Science, Stanford University, Stanford, CA 94305

general enough to accommodate a large class of metamorphic robots with different geometry and motion constraints. It is essentially an extension of the model developed in (Chirikjian et. al, 1996), with the emphasis on motion constraints. The distributed reconfiguration control algorithms in this paper are developed for the **Proteo** model.

The challenge for distributed control for self-reconfiguration is that each module must decide its next move based on the desired final configuration and only its local state, which includes delayed or incomplete global information obtained via communication from other modules.

The key behind the algorithms presented is the "ordered-goal" constraint, i.e. locations in the final configuration are filled in a pre-defined globally known partial order. The use of the "ordered-goal" constraint ensures some kind of monotonicity or stability of the distributed system and also greatly reduces the communication time for each module to obtain the global information it needs about the system.

Three algorithms based on the "ordered-goal" constraint are developed. One is distance-based, where modules move, generally, toward the closest unfilled goal. Another is heat-based; it simulates a heat flow from goals to moving modules, and modules move, generally, along the temperature gradient. The third is a hybrid of those two, which greatly reduces local minima compared to the individual methods. All of these distributed algorithms are parallel, both in the sense that all modules compute in parallel and in the sense that several modules may move at the same time. All the algorithms work for any metamorphic robotic system that fits the definition of the **Proteo** model.

The control algorithms were tested in a simulation environment. While none of the methods guarantees to fully reach final configurations, our testing results show that the hybrid method does outperform the other two in terms of the 100% goal completion. Further, the observed configuration times scale approximately linearly over an order of magnitude range of the number of modules.

The simulation environment, written in **Java** with **Java 3D,** supports the general **Proteo** model with motion constraints and allows one to "plug and play" any control algorithm derived from a base control structure. The system has been used to study biologically inspired systems and emergent structures (Bojinov et. al, 2000).

1.2 Related Work

Much work on metamorphic robots has been done in the past five years, in both design and motion control/planning. Existing 2D metamorphic robots include Hexagonal Modules (Pamecha et. al, 1996), Fractum (Murata et. al, 1994) and 2D Self-Organizing Collective Robots (Hosokawa et. al, 1998). Existing 3D metamorphic robots include 3D Self-Reconfigurable Structure (Murata et. al, 1998), Robotic Molecule (Kotay et. al, 1998), and Crystalline Modules (Rus and Vona, 1999, 2000).

Both motion planning and distributed control of such metamorphic robots have been studied.

For motion planning, (Pamecha et. al, 1997) presented a technique of simulated annealing to drive the reconfiguration process with configuration metrics as cost functions. Performance associated with different configuration metrics was also analyzed. Since finding an optimal motion sequence is believed to be computationally intractable, (Chirikjian et. al, 1996) presented methods for computing upper and lower bounds for such sequences, which can be used as guidance for motion planning.

For distributed control, (Murata et. al, 1994) used a diffusion-like process with a local type fitness measure to obtain a movability strategy, while the actual motion is random. The randomness of motion makes convergence of the system problematic. A distributed simulated annealing approach was described in (Murata et. al, 1998), however, the system performed well only when there was a small number of modules (up to 20). Even though there is no fundamental difference, most of the planning and control algorithms previously described have only one module move per time step.

In general, there have been no algorithms or control strategies that guarantee 100% completion of arbitrary goal configurations with motion constraints. Two approaches have been investigated in literatures. One is to use meta-modules, i.e., a collection of modules as one unit to move around. Examples of this approach were discussed in (Rus and Vona, 1999) and (Nguyen et. al, 2000). Another is to restrict the space of configurations. (Nguyen et. al, 2000) presented a theoretical proof for a 2D **Proteo** model, which can be extended to 3D as well.

Most planning and control strategies are for self-reconfiguration with the final configuration specified in some form. Emergent structures is another interesting approach, i.e., final configurations are not specified explicitly but are emergent as the result of applying local control rules. (Hosokawa et. al, 1998) developed

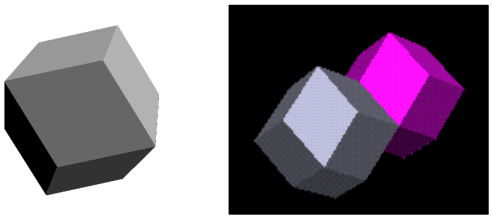
control rules for their 2D Self-Organizing Collective Robots to traverse a vertical plane. (Bojinov et. al, 2000) generated several biologically inspired control laws to "grow" stable structures for a 3D **Proteo** model. With this approach, even though the specific shape of the resulting structure is non-deterministic, the emergent structure has the desired functionality.

2. The Proteo Model and Motion Constraints

Proteo is a class of metamorphic robots that is composed of uniformly shaped modules that normally occupy individual spaces on a uniform discrete grid or lattice. **Proteo** treats motions from one grid position to another as discrete steps. A single discrete step of a **Proteo** module is constrained to its open neighbor sites in the lattice, with the support of one of its connected modules.

2.1 Example

An example of a **Proteo** module has the shape of a rhombic dodecahedron (RD), a 12-sided **dual uniform** polyhedron that is a kind of 3D analog of a hexagon (see Fig. 1(a)). An RD module can move itself by rolling around one of its edges shared with another module (see Fig. 1(b)).



(a) (b)

Figure 1. (a) A rhombic dodecahedron (RD)
(b) Two RD's with two faces mated

The RD shape has several good properties:

1. An RD is isohedral (i.e., all faces are alike). This isohedral property simplifies manufacture. One advantage to repeating modules is the ability to batch fabricate, minimizing the unit cost. In addition, each module can be made of repeating parts. For example, each polyhedron is made of faces, each face made of edges and each edge made of vertices. If the polyhedron is an isohedron with uniform faces like the rhombic dodecahedron, then batch fabrication can be applied at the face level and the edge level as well.

2. Modules of RD shape are packed so they fill space with minimal gaps. When two faces of two identical RD's are aligned and pressed together (mated) as in Figure 1(b), a rotation about one of the shared edges will result in another set of faces aligning. All subsequent rotations will have the same result. This property is also true for cubes and all regular polyhedra, however it is not true for all isohedra.

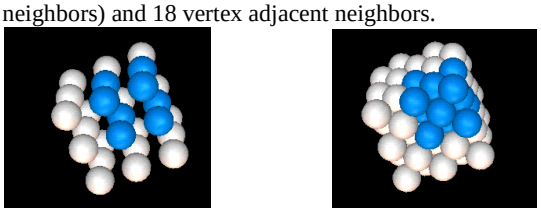
3. Like hexagons, an RD module requires only a single simple rotational motion to move to an open neighbor site; the rotation about any edge of an RD module from one site to another is always exactly 120 degrees. The reason for this simplicity is that, when packed, every edge is formed by the adjacency of at most three RD's. When there are three RD's around one edge, no motion can occur about that edge — the 360-degree space is filled. When there is one RD, no motion can occur either since there must be another RD to be moved over for support. Thus motion can only occur when there are exactly two RD's. This uniformity of the edge relationships is the key to the simplicity of the design of a mechanism that allows rotations about edges.

This is a promising example of the **Proteo** model. However, the actuation design for the RD shaped module turns out to be much more difficult than expected. The algorithms presented in this paper are tested in a simulation environment with RD shaped modules. However, the same algorithms shall work for the general **Proteo** model without committing to any particular shape.

2.2 Proteo Spaces

Proteo robot modules reside in a grid space. A *grid space* consists of a regular uniform lattice (or "array"), with each *site* ("cell") either empty or occupied by a module. For simplicity, obstacles are not yet considered. Figure 2 shows two typical types of 3D-grid spaces that are described by the way uniform objects pack. Formally, let $\boldsymbol{Z}$ be the set of integers; then the 3D-grid space described by the *simple cubic packing* is $\boldsymbol{Z^3}$. Similarly, the space described by the *face centered cubic packing* is a subset of $\boldsymbol{Z^3}$ with the sum of three coordinates (x+y+z) even. A *neighborhood* of a site is defined to be a set of adjacent

sites. For example, for the simple cubic packing, there are 6 face adjacent neighbors, 18 edge adjacent neighbors, and 26 vertex adjacent neighbors; for the face centered cubic packing, there are 12 face adjacent neighbors (which are also the 12 edge adjacent neighbors) and 18 vertex adjacent neighbors.



(a) (b)

Figure 2. (a) Simple cubic packing
(b) Face centered cubic packing

RD modules naturally reside in the 3D-grid space that is described by the face centered cubic packing. And for the RD, two sites are neighbors if and only if they are face adjacent; there are a total of 12 neighbors for each site.

*2.3 **Proteo** Modules*

A ***Proteo** module* is an electromechanical device; with embedded sensors, actuators and controllers. Two modules in a grid space can be *connected* if their sites are neighbors, which can be face, edge or vertex adjacent depending on the shape of modules or hardware design. Note that even though this definition does not imply that two modules are rigidly connected, in implementation it is normally the case. Communication will be established between two modules when they are connected. The physical position of a module can be uniquely described by its site in a grid space, e.g., its Cartesian coordinates. This assumes that the module's orientation doesn't matter; either the module is symmetric or its orientation does not change by motion. A module can *move* itself to one of its open neighbor sites by the support of another connected module, e.g. rolling over the other module, at a discrete time step.

A ***Proteo** robot* consists of a set of connected, identical, (in the sense of both hardware and software), modules in a grid space. A *configuration* of a **Proteo** robot is the set of sites occupied by the modules in the grid space. A **Proteo** robot changes its configuration by a series of module motions. The **Proteo** robots fit the description of "Digital Robots" (Walker and Cavallaro, 1999).

2.4 Motion Constraints

When is a module in a **Proteo** robot able to move to one of its open neighbor sites? Different mechanical designs lead to different answers to this question. However, there is a set of common properties:

1. A module moves relative to another module, denoted as a *parent.* There must exist a neighboring parent module to move.
2. A module may only be attached to its parent while moving. It may not carry other modules while moving.
3. The motion of the module must not collide with any other module.
4. The entire group of modules must remain connected after the motion (assume the modules are connected initially).
5. There is a *fixed base* module, which does not move.

Note that gravity constraints have not yet been incorporated, i.e., it is assumed that the connection between modules is strong enough to hold modules together in any configuration, or that gravity is negligible (such as in space).

The first property restricts the way modules move around each other. The second property greatly simplifies planning/control problems and eases mechanical design. The last two properties stem from the assumption that one power source supplies power to all modules through connected modules from the fixed base module. The extra fixed base constraint makes the self-reconfiguration problem harder. However, the control algorithms developed later are applicable to **Proteo** robots without the fixed base constraint as well.

Property three, which varies from system to system, is a *blocking constraint*. At one extreme, if a module is entirely surrounded by all its neighbor modules, then that module cannot move. At the other extreme, if a module is only connected to a single module, then it is free to move to any of the neighboring sites supported by the other module. This blocking constraint describes the situation of a module preventing other modules from moving into or out of a position. Given a **Proteo** model, *blocking constraint*s are defined as follows. Let ***M*** be a site in the grid space, ***N*** and ***B*** be neighbor sites of ***M***, and ***P*** be a neighbor site of both ***M*** and ***N*** (see Fig. 3 for a 2D illustration). The triple ⟨***N, P, B***⟩ is

a *blocking relation* for ***M*** if and only if a module at site ***B*** blocks the motion of a module at site ***M*** toward the site ***N*** with the support of a module at ***P***. In other words, the relation says that a module at site ***B*** stops a module at site ***M*** from moving out of its current location if it tries heading toward site ***N*** with the support at site ***P***. Note that blocking relations are defined *locally* and it is always the case that ⟨***N, P, N***⟩ is a blocking relation, namely, any module at a neighbor site *N* blocks its motion to *N*.

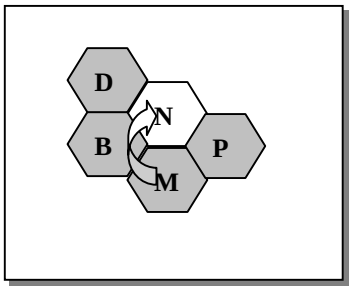


Figure 3. A blocking relation for 2D hexagon modules

Motion constraints are symmetric, i.e., if a module at site ***M*** cannot move to its open neighbor site ***N***, a module at site ***N*** cannot move to its open neighbor site ***M***. A *blocking constraint* can be defined via blocking relations: A module at site ***M*** can move to its neighbor site ***N*** with the support of a module at site ***P*** if:

1. [**Leaving Constraint**] There is no module at site ***B*** such that ⟨***N, P, B***⟩ is a blocking relation for ***M***, and
2. [**Arriving Constraint**] There is no module at site ***B*** such that ⟨***M, P, B***⟩ is a blocking relation for ***N***.

For rigid 2D modules in Figure 3, a module at site ***M*** cannot move to site ***N*** if there is a module at site ***B*** (violates the leaving constraint) *or* if there is a module at site ***D*** (violates arriving constraint). Note that the blocking constraint is local in the sense that if a module at ***B*** blocks the motion from ***M*** to ***N***, ***B*** is a neighbor to either ***M*** or ***N***, i.e. ***B*** is a neighbor or a neighbor of the neighbors of ***M***. A site ***N*** is said to be within the *second-order neighborhood* of a site ***M*** if and only if ***N*** is a neighbor of ***M*** or ***N*** is a neighbor of the neighbors of ***M***. Thus, a blocking site ***B*** of ***M*** must be within the second-order neighborhood of ***M***.

Given any pair of neighbor sites ⟨***N, P***⟩ of ***M***, if ***n*** is the number of sites ***B*** such that ⟨***N, P, B***⟩ is a blocking relation for ***M***, the system is said to have an ***n-side constraint***. For 2D rigid hexagon modules in Figure 3, such as Fractum (Murata et. al, 1994), the system has the 3-side constraint. However, if the hexagon is flexible, such as the one designed in (Pamecha et. al, 1996), i.e. deforming itself during motion, the system would only have the 1-side constraint. Similarly, RD-shaped **Proteo** can have a 7-, 5-, 3-, or 1-side constraint depending on how the mechanical system is designed. The 7-side constraint corresponds to a fully rigid RD shaped module. The 5-, 3-, and 1-side constraints correspond to different levels of deformation of the RD shape during motion. While this paper will not discuss the mechanical design for various blocking constraint types, it is clear that the mechanical design becomes more difficult as ***n*** goes down, but the control system becomes easier. At the extreme with the 1-side constraint, a module can move to any open neighbor site by deforming and squeezing between any neighboring modules that would block a rigid RD module.

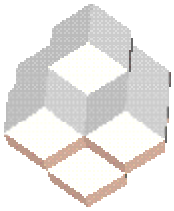
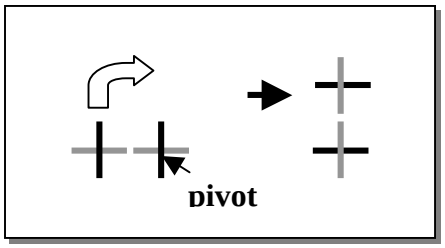


Figure 4. An immobile configuration of 6 RD.

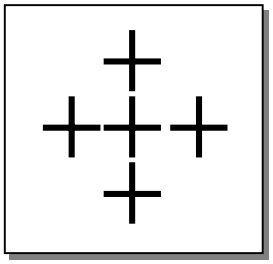
A subset of modules in a configuration is called *immobile* if and only if no module in that subset can move independent of the surrounding structure of the subset. A blocking constraint for a **Proteo** model is *satisfiable* if and only if there do not exist immobile subsets in any configuration. Blocking constraints that are not satisfiable can lead to unsolvable reconfiguration problems. For example, the 3-side constraint for the 2D hexagon model is satisfiable. However, the 7-side constraint of the RD model is not satisfiable: if six RD's are brought together at a common vertex as shown in Figure 4, no RD can move. Since motion constraints are symmetric, the inverse is also the case: modules cannot move into this subset of any configuration. This subset is encompassed within many possible configurations, most notably any solid configuration with thickness greater than three RD modules. This result greatly restricts the set of configurations that can be built by the rigid RD-shaped **Proteo** model. However, if a little deformation is allowed during motion, resulting in the

5-side constraint, there are no configurations that are immobile due to blocking constraints alone.

However, satisfiable blocking constraints do not imply that all configurations are mobile. An example of a satisfiable blocking constraint is Murata's 3D Self-Reconfigurable Structure (Murata et. al, 1998), which is in the 3D-grid space $\boldsymbol{Z^3}$ described by simple cubic packing. A module can only move to an edge adjacent neighbor site using a face adjacent neighbor as the pivot support as in Figure 5(a). Note that even though the pivot (parent) module rotates 90 degrees, it is considered stationary in this model since it does not move to another position in the grid and rotations result in functionally identical orientations due to its symmetry. The system has a 2-side constraint. Even though the blocking constraint is satisfiable, an immobile configuration projected in 2D is shown in Figure 5. However, it is not an immobile subset since



(a)



(b)

Figure 5. (a) A step move in Murata's 3D Self-Reconfigurable Structure (b) Immobile configuration

adding any module to the subset results in a mobile configuration.

Note that in general, motion constraints are *not* local. The fourth property, connectivity, is one example of a global constraint. A system with a satisfiable blocking constraint can also have immobile configurations due to a combination with other motion constraints, such as connectivity. For example, (Nguyen et. al, 2000) showed a configuration of 2D hexagon with 3-side constraint in an immobile state, due to the combination of the blocking constraint, the fixed base constraint and the connectivity constraint.

If more than one module is allowed to move at one time, there are further motion constraints:

1. No two modules move to the same open site.
2. Parents cannot move while supporting the motion of other modules.
3. No module becomes a blocking module for a moving module.

A *single step motion* of a **Proteo** robot is a set of simultaneous single step moves that satisfy all the motion constraints.

2.5 Reconfiguration Problems

The *reconfiguration problem* for a **Proteo** robot is defined as follows: Given an initial configuration ***I*** and a final configuration ***F***, find a series of single step motions that leads from ***I*** to ***F***. Even though related, two different kinds of reconfiguration problems, reconfiguration motion *planning* problems and reconfiguration motion *control* problems, shall be defined. For motion planning, the inputs are the initial and final configurations ***I*** and ***F***, and the output is a series of single motion steps that leads from ***I*** to ***F***. For motion control, the inputs are the current and final configurations, as well as the *state* of the control, and the output is a single step motion that moves *towards* the final configuration.

At one extreme, a motion planner may produce the whole series of steps off-line and a motion controller may follow the steps open loop. At the other extreme, a motion controller may determine each of the steps sequentially on-line. Between these extremes, a motion planner can be used inside a motion controller to obtain a partial or complete sequence to the final configuration in order to assure the best move for the next step (e.g., optimal or guaranteed success). The controller then follows the first few steps until a new plan is produced.

On the other hand, a motion controller can also produce a motion plan by recording all the steps to the final reconfiguration. However, it is most likely that such a "plan" is not optimal, i.e., within the minimal number of steps or moves. Nevertheless, as shown by many people, (Latombe, 1991) for example, optimal planning of the motion of ***n*** robots, or in this case modules, moving at the same time from one

configuration to another, is intractable, as the search space is exponential in ***n***.

This paper focuses on motion control, in particular, distributed motion control of the reconfiguration problem. The purpose of studying distributed control is to push the idea of homogenous systems to an extreme, so that all the modules not only have the same hardware, but also have the same software.

3. Distributed Control of Reconfiguration

In distributed control of **Proteo** robots, all modules have an identical controller and each controller *decides* where to move, depending on its current site, its current state, and states of neighbors determined via local communication. This includes limited sensing to determine collision and disconnection detection.

Reconfiguration problems are hard. Distributed control for self-reconfigurations is harder. There are several specific issues for distributed control:

1. **Stability**: The control law in each controller locally has to confirm to some type of global stability. The system should stop motion when its final configuration is reached.
2. **Local minima**: The problem of local minima exists in most local control laws. Extra care must be taken to minimize the factor of local minima, such as adding randomness or turbulence.
3. **Overcrowding**: Modules sometimes overcrowd in an area, blocking the motion of each other. Extra communication has to be established to alleviate this situation.

This section defines the formal model for this class of control, and develops a type of distributed control with a global goal ordering. Three instances of such control are illustrated and their properties are discussed. Finally performance results of these control algorithms are shown.

3.1 Control Model

A distributed control model for **Proteo** robots is defined as follows:

1. All modules have identical controllers.
2. Each controller knows the final configuration, its own site and has a set of states including information about the global configuration.
3. At each time step, two modules can exchange state information when they are connected.
4. Each controller decides where to move according to its local information and its incomplete or delayed global information.
5. If a move is not achieved due to sensed violation of global motion constraints, the controller will be notified (via sensors) and will be allowed to revise its output as many times as needed.

The sensed motion constraint violations include:

1. There is a module that is moving to the same destination site.
2. Its parent module is moving.
3. There is a moving module that becomes a blocking module.
4. The module to be moved will divide the system/robot into two disconnected parts.

Condition 1,2 or 3 happens only if more than one module moves at the same time.

In this model, communication can be used to build the global state of the configuration as messages may be passed from module to module. If nothing is moved for more than ***N*** steps, where ***N*** is the number of modules, and every module communicates its site information and passes the site information it has received, every module is able to know the sites of all the other modules. In other words, if each module communicates ***N*** times before making a decision for moving, each module would have the global state information for making that decision. In real situations, modules would not necessarily be restricted to communicate only once per motion step as a mechanical move is normally much slower than an electronic data move. However, letting ***N*** modules communicate ***N*** times when ***N*** is very large would still slow down the reconfiguration process. Considering that every step of communication time takes ***O(N)*** since the size of the information is ***O(N)***, ***N*** steps would take $O(N^2)$. In practice, the number of communication steps for a move shall be set to a constant ***n*** for ***n*** much less than ***N***, depending on the time for making one step motion mechanically.

For each control cycle, the controller works as shown in Figure 6. The “Reset” phase is used to synchronize all the initial information among distributed controllers before communication. During the “Communication” phase a module exchanges information with its connected modules. There can be ***n***, where ***n*** is greater than or equal to 1 and less than or equal to the number of modules, number of communication steps before a mechanical move. Then, in the “Decision” phase it computes the next moving

direction according to its current information. During the "CanMove" phase, it checks, via sensors, whether or not motion constraints are satisfied. The module enters the "Revise" phase and re-computes the next possible move when a move cannot be made due to unforeseen (or global) motion constraints.

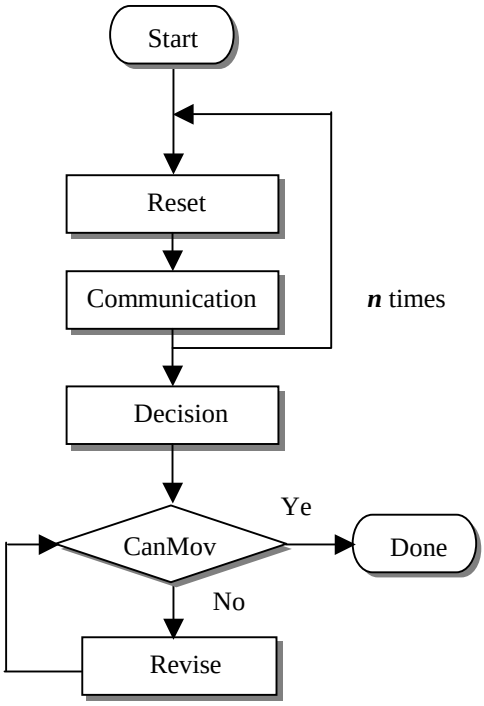


Figure 6. Flowchart of each distributed controller for one step motion

3.2 Goal Ordering

A *goal* is a site in the final configuration. The distributed control for the reconfiguration problem presented here is based on *goal ordering*: a partial order defined on a grid space that will be used to determine the order for filling goals of any final configuration. Let < be a partial order defined on the grid space. If L_1 and L_2 are two sites in the grid space and ($L_1 < L_2$), L_1 will be filled before L_2, given L_1 and L_2 are goals; L_1 *has lower order* (or higher priority) than L_2 and L_2 has *higher order* (or lower priority) than L_1.

There are two purposes for goal ordering:

1. To enforce some *order* for filling the goals so that once a module fills a goal in the order, it can stay there forever without blocking other modules moving to the rest of the unfilled goals.
2. To enable local *reasoning* about global information so that communication can be kept minimum.

Incorporating the blocking constraint and the fixed base constraint, an ordering defined on the grid space is called a *goal ordering* if and only if it satisfies the following conditions:

1. The site of the fixed base belongs to the set of sites that have the lowest order.
2. For an ***n***-side constraint **Proteo** model, at least ***n*** neighbor sites have higher order than the given site.
3. There must exist a parent ***P*** and a neighbor site ***N*** pair $\langle N, P \rangle$ of any given site ***M*** whose set of blocking neighbors is a subset of neighbor sites that have higher order than ***M***.

An example goal ordering follows for an RD shaped **Proteo** model with the 5-side constraint. An ordering can be defined via a single-valued function ***H*** as follows: let $H(L) = |y_L - y_0 + z_L - z_0|$ where (x_0, y_0, z_0) is the base coordinates, and $L_1 < L_2$ if and only if $H(L_1) < H(L_2)$. Without loss of generality, we can assume that the fixed base is at the origin, $(0, 0, 0)$, and L is at (x, y, z), so that $H(L) = |y + z|$.

Proposition 1. *The ordering defined above is a goal ordering for RD with the 5-side constraint.*

Proof: For any site L with coordinates (x,y,z), there are 12 neighbor sites at $(x+1, y+1, z)$, $(x-1, y+1, z)$, $(x-1, y-1, z)$, $(x+1, y-1,z)$, $(x+1, y, z+1)$, $(x, y+1, z+1)$, $(x-1, y, z+1)$, $(x, y-1, z+1)$, $(x+1, y, z-1)$, $(x-1, y, z-1)$, $(x, y-1, z-1)$. If L is the fixed base, $H(L) = 0$, therefore, the fixed base has the lowest order. Since $H(L)$ is an absolute value, there are two cases for $H(L) > 0$. For $y + z > 0$, $(x+1, y+1, z)$, $(x-1, y+1, z)$, $(x+1, y, z+1)$, $(x-1, y, z+1)$ and $(x, y+1, z+1)$ is the set that has higher order. These sites correspond to the blocking neighbors of four parent and neighbor pairs, one of which is $\langle N=(x+1, y-1, z), P=(x+1,y, z+1)\rangle$. Similarly, for $y + z < 0$, $(x+1, y-1, z)$, $(x-1, y-1, z)$, $(x+1, y, z-1)$, $(x-1, y, z-1)$ and $(x, y-1, z-1)$ is the set that has higher order and are the blocking neighbors of four parent and neighbor pairs, one of which is $\langle N=(x+1,y, z+1),$

P=(x+1, y-1, z)⟩ (note ***P*** and ***N*** are swapped from the previous case). For any given site ***L*** with ***H(L) = 0***, both cases apply and there are $10 \geq 5$ neighbor sites with higher order than the given site. For all other cases (***H(L) > 0***), there are exactly 5 neighbor sites with higher order than the given site. □

Proposition 2. *If **G** is a goal ordering for an **n**-side constraint, **G** is also a goal ordering for an **m**-side constraint for all **m** < **n** whose set of blocking relations are a subset.*

Proof: If ***G*** is a goal ordering for an ***n***-side constraint, it satisfies the conditions for the goal ordering. If ***m*** < ***n***, all the conditions are still satisfied for an ***m***-side constraint if its blocking relations are a subset. □

Readers are encouraged to verify that there is no goal ordering for the RD with the 7-side constraint. In general, the following proposition holds.

Proposition 3. *A goal ordering for a **Proteo** model exists if and only if its blocking constraints are satisfiable.*

Proof: A goal ordering imposes a constraint on the order of motions to fill a configuration. If a blocking constraint is satisfiable, then there exists a goal ordering, since one can always start with the largest configuration with all the sites occupied by modules and define an order by assigning the highest order to one of the modules that can move, removing this module and assigning the second highest order to one of the modules that can move in the remaining configuration, and so forth. Conversely, if there is a goal ordering, there does not exist an immobile subset in any configuration due to the blocking constraints, since a module with the highest order, which has no blocking neighbors with respect to a parent, can always move, if the parent exists. □

A goal is called *filled*, if and only if a stabilized module occupies it. A goal is called *constrained* if filling it would block other goals from being filled. Specifically, goal ***G*** is constrained if and only if there is an unfilled goal ***G'***, ***G'*** < ***G***, which is within the second-order neighborhood of ***G***. A goal that is not constrained is called *unconstrained*. The unconstrained property of a goal can be calculated using the goal ordering and the information on the filled goals. Note that "filled" is a global property, but "unconstrained" is a local property, dependent only on the state of nearby sites.

A distributed controller based on the goal ordering can be constructed as follows: each controller has an array named "filled" and an array named "unconstrained", indexed by goals in the final configuration. A module will no longer move as soon as it comes to an unconstrained goal site. An element in the "filled" array is set as soon as the corresponding unconstrained goal is filled with a module. The "filled" arrays are propagated every step via communication by updating connected neighboring modules. In addition, more filled goals can be deduced from the goal ordering, i.e., whenever a goal is filled, all its neighbor sites with lower ordering must also have been filled. At any time, the "filled" array is a module's internal snapshot of the global configuration. At the same time, elements in the "unconstrained" array will be deduced using the goal ordering and the states of the "filled" array. The procedure implies the following consequences:

1. Both "filled" and "unconstrained" arrays are monotonic in time, i.e., when an element is set, it is forever set.
2. Both "filled" and "unconstrained" arrays are conservative, i.e., if an element in the "filled" (or "unconstrained") array is set, it is guaranteed that the goal is actually filled (or unconstrained).
3. A "filled" goal is an "unconstrained" goal, i.e. the "filled" sites are a subset of the "unconstrained" sites.

The system has a set of control modes that are listed in the next section; one of the modes is "goal-reached" which indicates that the module has occupied an unconstrained goal and will no longer move. If the module is not in the "goal-reached" mode, the module will decide how to move by ordering the set of open neighbor sites according to some "neighbor ordering" criteria. The lowest in the ordering that satisfies local motion constraints will have the highest priority and be chosen first, and the rest will be tried out in order if revision is required.

The algorithm stops when all the elements in the "filled" array are set. Because the "filled" array is conservative and monotonic, the distributed control based on the goal ordering is "stable" in the sense that the number of "unfilled" goals will not increase. However, it does not guarantee that the final configuration will be achieved.

3.3 Goal-Ordering based Control Algorithms

This section first discusses two types of methods corresponding to two different neighbor-ordering

strategies. The properties of the algorithms are analyzed and a hybrid method is then proposed.

3.3.1 Distanced-based Method

The distance-based method uses the "distance" to unconstrained unfilled goals as the major measure for ordering open neighbor sites. Euclidean distance is used in the algorithm. However, any other distance metric should work about the same. Performance may vary by using different metrics (Pamecha et. al, 1997). Each module has a variable "target", which is the closest unconstrained unfilled goal to the current module according to the distance measure. Note that two or more modules can have the same target. Once the target is filled, a new target will be chosen.

Some heuristics have to be used to solve the problem of overcrowding, i.e., too many modules are around a target site, blocking each other from the target site. The problem is solved using the principle of "competition and cooperation". Competition takes place when modules are far away from the target site, and cooperation takes place when more than one module is a neighbor of the same target. By reserving the target for only one of these modules and redirecting the rest to other targets, overcrowding will be relieved.

In addition to the "goal-reached" mode, there are four other modes in this method: namely, "no-goal", "reserve-goal", "assist-goal" and "normal-goal".

1. "**no-goal**": every module is in "no-goal" mode initially, and will become "no-goal" again whenever its target is filled, or reserved by other modules. In this mode, a new target is chosen from the closest unconstrained unfilled goals which have not been reserved by others at that time. If no such target exists, i.e., all the unconstrained goals are either filled or reserved by others at the time, a dummy goal, which is outside of the final configuration, will be chosen. The dummy goal can be random, or any site just outside of the edge of the goal configuration. In our experiments, we first find M as the goal site with the largest H, where H is defined as $|y+z|$. We then use $D = M + (0,4,4)$ if $(y+z)>0$ and $D = M + (0,-4,-4)$ if $(y+z)<0$. This helps to relieve overcrowding situations by allowing modules with nowhere to go to move away, so that the goals can be filled by modules that have reserved them.

2. "**reserve-goal**": a module at site M will reserve its target goal T, if and only if,
   a) T is its neighbor,
   b) $T < M$, and
   c) there exists another goal G, a neighbor of T, with $G < T$.

   If a module satisfies these three criteria, it records the time that it reserved the goal. The module then keeps this reservation for some steps, or until it communicates with a neighbor that reserved the same goal at an earlier time. This reservation information is then propagated from its neighbors to other modules moving toward this target. Any other module moving towards the same goal is forced to choose a different goal. By forcing other modules to choose other goals, the module making the reservation should have more room to move. According to the goal ordering constraint, G must have been filled. If G is a neighbor of M, the module can then use G as its parent, rolling over G to fill is target T.

3. "**assist-goal**": if a module M has reserved a target T, but there is no filled goal G that is a neighbor of M, M will raise an assistance-required flag. If a connected module is also moving toward T, then instead of choosing another goal, it assumes the "assist-goal" mode and assists the reserving module M by moving to a neighbor site P satisfying $P < T$. The reserving module can then lower the assistance-required flag and roll over the assistant to reach its goal.

4. "**normal-goal**": the system is in "normal-goal" mode if it has a target. The mode will switch to "no-goal", "reserve-goal", or "assist-goal" whenever the corresponding conditions are satisfied.

The transition between these modes is depicted in Figure 7.

For all modes, except "goal-reached", the control will decide which open neighbor site to move to, according to some ordering with respect to its target. Neighbor sites are ordered in terms of its connectivity and its distance to the target. A site that is a neighbor of a filled goal or is a neighbor of at least two modules is ranked with a higher priority. This is called the *connectivity* ranking. If this ranking is the same for two neighbor sites, the distance to the target will be used for ranking; the one with shorter distance is ranked with a higher priority. If both factors are the same, then the two sites are ordered randomly. The modules tend to stay together rather than spread out in long thin chains toward goals, due to the connectivity criterion. The lack of long chains results in a significant decrease

in the number of local minima for the distance heuristic.

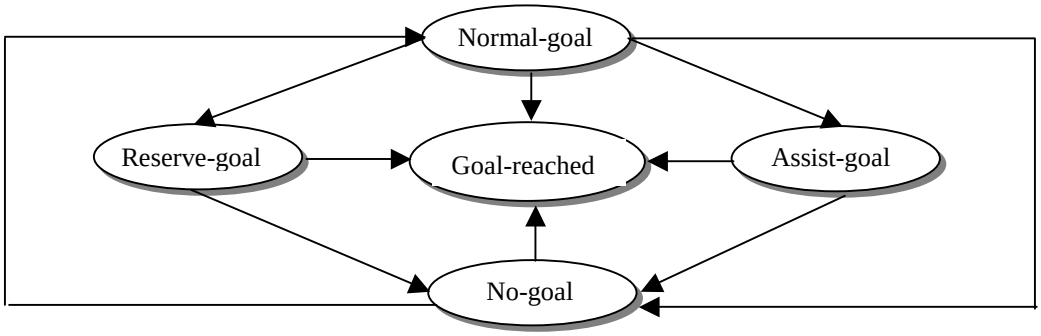


Figure 7. State-transition for distance-based control

3.3.2 Heat-based Method

The heat based method uses a simulated "temperature" as a measure for ordering open neighbor sites, in which unfilled unconstrained goals are heat sources and non-goal-reached modules are heat sinks. Unfilled unconstrained goals produce heat (increase temperature) at every step and non-goal-reached modules consume heat (decrease temperature) at every step. Heat sources propagate portions of the heat through contacting modules. All modules propagate heat to their neighbor modules. This propagation is heat preserving, i.e., the heat added to a neighbor module is subtracted from the given module.

In this control, each module has a state variable "temperature" which is set to 0 initially. In addition, a "goal temperature" array is updated at every step locally. During the "Reset" phase (see Figure 3.1), each module that is not yet "goal-reached" will *decrease* its temperature by one unit, and each unfilled unconstrained goal will *increase* its temperature by one unit. During the "Communication" phase, heat is propagated through modules. Formally, let ***T(t)*** be the temperature of a given module at time ***t***, and ***n*** be the number of neighbor sites for the grid space (12 in the case of RD):

$$T(t+1) = T(t) + \sum_{i \in neighbor-modules} \frac{T_i(t) - T(t)}{n} + \sum_{g \in neighbor-goals} \frac{T_g(t)}{n}$$

where $\boldsymbol{T_i}$'s are temperatures of connected modules and $\boldsymbol{T_g}$'s are temperatures of neighboring unfilled goals. The result is, the modules closer to the unfilled unconstrained goals are warmer than the modules far away. During the decision making phase, the temperatures of open neighbor sites are estimated by averaging the temperatures of connected modules. Open neighbor sites are ordered with the higher temperature ranked higher priority. If the temperatures for two open neighbor sites are the same, the two sites are ordered randomly.

3.3.3 Problems and Solutions

Neither of the methods discussed above *guarantee* the reachability of final configurations, even though in practice, they do reach final configurations most of the time. Some of the unsuccessful cases are due to the properties of final configurations and some are inherent to the properties of the methods.

Final configurations that are too dense (e.g. a big solid ball) or too sparse (e.g. a hollow ball of one module thick) are hard to achieve. For the dense case, modules tend to suffer from overcrowding. For the sparse case, there are not enough modules to support rolling over each other.

Another type of final configuration that is hard to achieve has structures with partitioned (e.g., a hollow ball, or any structure that encloses spaces) or concave (e.g., a cup) spaces. For example, a moving module inside the bottom of a cup will be stuck and unable to fill a goal outside the bottom of the cup (see Fig. 8(a)). Based on the assumption that the final configuration is known to every module, the problem can be solved by assigning exclusive areas (such as the inside of a hollow ball) outside the final configuration and keeping modules from moving into these areas in the control strategy. This strategy no longer works if there are branches to be filled inside a hollow ball.

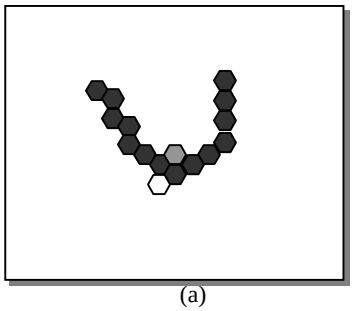
There are also final configurations with "black holes". A site that is not a member of a configuration is called an

exterior site of the configuration if and only if it is a neighbor of a site in the configuration. An exterior site of a final configuration is a *black hole* if a module in the site cannot move when its neighbors are filled. An example of a final configuration with a black hole is a plane with an interior site removed (see Fig. 8(b)). Final configurations with black holes may trap moving modules and keep them from reaching their final goals. The strategy of assigning exclusive sites can be used if black holes are identified a priori.

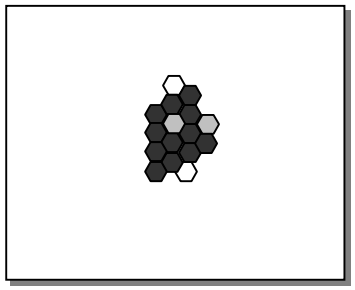
Because the decision making in these methods is local in both space (within connected modules) and time (current state), local minima can occur. To alleviate this, two strategies have been taken for ordering open neighbor sites:

1. **added randomness**: open neighbor sites are ordered randomly given that the other criteria are the same.
2. **imposed noise**: the best open neighbor site is skipped and the second best neighbor site is chosen from time to time to avoid being trapped in the same situation.

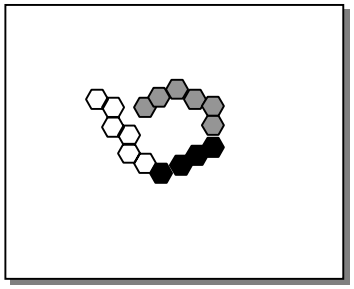
A significant difference between the distance-based method and the heat-based method is that the heat-based method considers the module motion along the surface of the configuration, while the distance-based method assumes modules can move freely in space. One unsolved problem with the distance-based control is the formation of long chains of modules where the target of the end module is past the end of the chain (see Fig. 8(c)). The chains may not be deformed and the system would be stuck without further progress. The problem is hard to solve within this method. On the other hand, the heat-based method solves the long chain problem naturally, since the ends of chains tend to get colder and colder while the body that contacts unfilled goals are getting warmer and warmer. Therefore, modules would move closer to the body. However, the heat-based method tends to be much slower, since the directions of the goals are blurred with local temperatures and propagation of temperatures has a large delay. In fact, it is hard for a module to find the right direction to move when it is far away from goals.



(a)



(b)



(c)

Figure 8. Hard situations for reconfigurations (a) goal with partitioned spaces (b) goal with black hole (c) long chain; where sites in black: goal-reached modules; in gray: moving module; in white: empty goal site

3.3.4 Combined Method

Based on these observations, a method combing both methods is developed. In the combined method, the control starts with the distance-based method, and switches to the other method when the system seems stuck. The key problem in this method is switching *simultaneously* in every module's controller without using some instantaneous global communication. For the combined method, every module has an extra state, named "stuck-time", which is used to estimate the number of steps since the last goal was filled globally. If a module occupies an unfilled unconstrained goal, and its "stuck-time" is less than a preset value $\boldsymbol{m}$, its "stuck-time" is reset to 0. In the "Reset" phase, the "stuck-time" is increased by one. During the "Communication" phase, the "stuck-time" of each neighbor is communicated and the smallest "stuck-time" among them is set to be the new "stuck-time". Thus, the minimum "stuck-time" is propagated to all the modules.

Let $\boldsymbol{D}$ be the maximum distance between two modules at any time. The following proposition holds.

Proposition 4. *If the "stuck-time" of one module is beyond* ***D*** + ***m****, the "stuck-time" of every module is beyond* ***D*** + ***m***.

Proof: The system is said to have *progress* if and only if there is a module whose "stuck-time" has just been reset to 0. Since ***m*** is the threshold that disallows a module to reset its "stuck-time" to 0, if there is no progress for ***m*** steps, the system will not have any further progress. ***D*** is the maximum number of steps for a new stuck-time (or any other piece of information) to be propagated to all modules since modules communicate once per step and communication spreads to all neighbors locally at each step. If the "stuck-time" of one module is beyond ***D*** + ***m***, then within the last ***D*** + ***m*** steps, there has been no progress; the last progress would have been propagated to every module before the last ***m*** steps. Therefore, the "stuck-time" of every module is beyond ***D*** + ***m***. □

From this property, ***D*** + ***m*** can be used as a threshold to switch from one method to another, since it guarantees that all the modules switch at the same time. In the worst case, ***D*** is equal to the number of modules ***N***. In practice, ***m*** can be set to any value greater or equal to 1.

3.4 Results and Performance

(Chirikjian et. al, 1996) presented an analysis on bounds for self-reconfiguration of (2D) metamorphic robots, in which the *maximal simply-connected overlap* is defined to be a maximum connected subset of the overlap between the current and the final configurations without loops. The same analysis for upper bounds can be carried out for **Proteo** robots, where "maximal simply-connected overlap" is replaced by "maximal constraint-free connected overlap". A set of modules is "constraint-free connected" if and only if any module in an exterior site of the set can move to any other exterior site via one or more steps, without violating motion constraints. Clearly the minimum such overlap is just the fixed base. Similarly, a lower bound on the total number of moves is given by an optimal assignment between the initial and final configurations, if there is only one module moving at a time.

These bounds only give references on how well a reconfiguration algorithm works in general. For distributed control with only local or delayed information, it is hard to guarantee the steps do not exceed the upper bound for all initial and final configurations. In fact, local minima may occur and the system may be stuck without being able to achieve the final configuration.

Various test cases are simulated, using the RD shaped 5-side constraint **Proteo** model with the initial configuration as a one module thick rectangular plane. Table 1 shows the number of time steps, one step communication per move, of the three methods, distance-based, heat-based and combined, for four types of final configurations, *flat disk*, *solid ball*, *hollow ball* and *cup*, with four different numbers of modules. The best algorithms for each case is marked in bond.

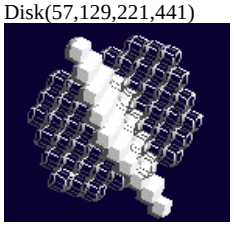
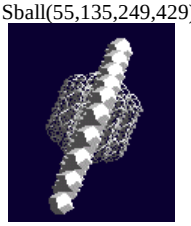
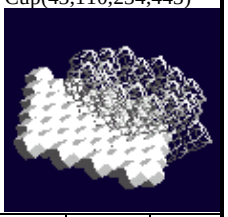
	Disk(57,129,221,441) 			Sball(55,135,249,429) 			Hball(42,114,302,450) 			Cup(43,110,234,443) 		
	D	**H**	**C**	**D**	**H**	**C**	**D**	**H**	**C**	**D**	**H**	**C**
Small	**116**	468	**116**	**125**	170	**125**	**65**	220	**65**	**62**	276	**62**
Small-Medium	**255**	1644	**255**	∞	**499**	574	**312**	3730	**312**	**361**	435	**361**
Medium-Large	**320**	4329	**320**	∞	1327	**807**	**952**	3186	**952**	**323**	2211	**323**
Large	**402**	∞	**402**	∞	∞	**2105**	**705**	3727	**705**	**537**	∞	**537**

Table 1. The number of steps for reconfiguration of various shapes and sizes, starting from a initial plane, where ∞ denotes either stuck or steps greater than 5000. The numbers in ( ) are actual number of modules for each final shape corresponding to Small, Small-Medium, Medium-Large and Large, respectively. **D**: Distance-based, **H**: Heat-based, **C**: Combined

The results show that the algorithms tend to take linear time with respect to the number of modules, if the best method is used. In the cases where the number of time steps is significantly large with respect to the linear curve, it is likely that more than 90 percent of goals are filled within less than 50 percent of time. Figure 8 shows how the overlap metric varies with time for the final configuration of a hollow ball with 302 modules, where the metric is defined by the percentage of non-overlap modules between the current and final configurations.

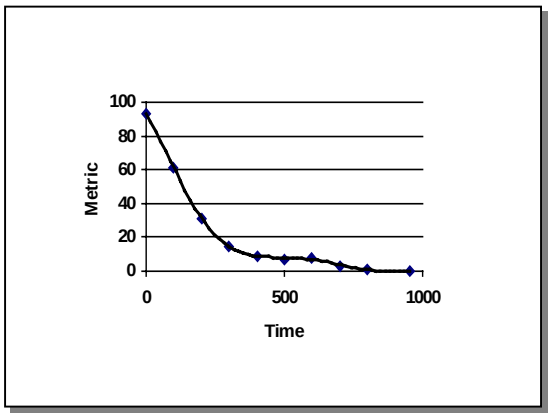


Figure 8. Reconfiguration process for a hollow ball

From the experiments, both the distance-based and the heat-based methods can get stuck: the distance-based method more likely gets stuck in a long chain, and the heat-based method gets stuck in a position where heat is balanced in all directions. Also, normally the heat-based method is slower than the distance-based method.

4. Conclusion and Future Work

We have presented a class of distributed control methods for 3D metamorphic modular robot reconfigurations. The methods apply to a class of metamorphic robots called **Proteo** whose characteristics have been formally defined. The properties of such methods are discussed and experimental results are shown. These methods have been tested in a simulation environment, which is available in the modular robots web site: http://www.parc.xerox.com/modrobots/Proteo/simulation.

4.1 Software Readiness

The control algorithms are developed and tested with the simulation environment. The control structure restricts the algorithm to use local information only. Even though the simulation does not run on "parallel processors", the algorithms are totally distributed and ready to be implemented for embedded processors.

4.2 Hardware Readiness

The prototype of the RD shaped **Proteo** modules is being designed. Due to the difficulty in obtaining the actuation mechanism, the first prototype will be "Digital Clay", with embedded sensing and communication, but with no actuation.

4.3 Future Research

The research on distributed control of 3D metamorphosis is far from finished. The following is a list of future work that is related to the content of this paper:

1. define or use goal ordering for blocking constraints that are not satisfiable,
2. choose different goal orderings for different types of initial and final configuration,
3. calculate efficiently upper and lower bounds of reconfiguration steps,
4. decompose a shape into a set of simple ones, design reconfiguration algorithms for each simple shape, and then combine the steps,
5. study special cases of configurations and control strategies that guarantees the goal achievement, and
6. incorporate gravity constraints into the model.

Acknowledgements

This work is funded by the Defense Advanced Research Project Agency (DARPA) under contracts #MDA972-98-C-0009 and #DABT630095C-025. Thanks also Arancha Casal for valuable comments and An Thai Nguyen for pointing out a subtle mistake in the paper.

References

Bojinov H., Casal A. and Hogg T. 2000. Emergent Structures in Modular Self-reconfigurable Robots. In *Proc. IEEE International Conference on Robotics and Automation*, San Francisco, CA. pp. 1734-1741

Chirikjian G. 1993. Metamorphic Hyper-Redundant Manipulators. In *Proc. JSME International Conference on Advanced Mechatronics*, pp 467-472.

Chirikjian G., Pamecha A. and Ebert-Uphoff I. 1996. Evaluating Efficiency of Self-Reconfiguration in a Class of Modular Robots. *Journal of Robotic Systems* 13(5):317-338.

Fukuda T. and Kawauchi Y. 1990. Cellular Robotic System (CEBOT) as One of the Realization of Self-Organizing Intelligent Universal Manipulator. In *Proc. IEEE International Conference on Robotics and Automation*, pp. 662-667

Hamlin G. and Sanderson A. 1996. Tetrobot modular robotics: prototype and experiments. In *Proc. IEEE/RSJ International Symposium of Robotics Research*, Osaka, Japan, pp. 390-395

Hosokawa K., Tsujimori T., Fujii T., Kaetsu H., Asama H., Kuroda Y. and Endo I. 1998. Self-Organizing Collective Robots with Morphogenesis in a Vertical Plane. In *Proc. IEEE International Conference on Robotics and Automation*, Leuven, Belgium, pp 2858-2863.

Kotay K., Rus D., Vona M. and McGray C. 1998. The Self-reconfiguring Robotic Molecule. In *Proc. IEEE International Conference on Robotics and Automation*, Leuven, Belgium, pp 424-431.

Latombe J.-C. 1991. *Robot Motion Planning*, Kluwer: Dordrecht, Netherlands.

Murata S., Kurokawa H. and Kokaji S. 1994. Self-Assembling Machine. In *Proc. IEEE International Conference on Robotics and Automation*, pp 441-448.

Murata S., Kurokawa H., Yoshida E., Tomita K. and Kokaji S. 1998. A 3D Self-Reconfigurable Structure. In *Proc. IEEE International Conference on Robotics and Automation*, Leuven, Belgium, pp 432-439.

Nguyen A., Guibas L. and Yim M. 2000. Controlled Module Density Helps Reconfiguration Planning. *In Workshop on the Algorithmic Foundations of Robotics*, pp TH15-TH27.

Pamecha A., Chiang C., Stein D. and Chirikjian G. 1996. Design and Implementation of Metamorphic Robots. In *Proc. ASME Design Engineering Technical Conference and Computers in Engineering Conference*, Irvine, California.

Pamecha A., Ebert-Uphoff I. and Chirikjian G. 1997. Useful Metrics for Modular Robot Motion Planning. In Proc. *IEEE Transactions on Robots and Automation* 13(4):531-545.

Paredis C. and Khosla P. 1993. Kinematic Design of Serial Link Manipulators from Task Specifications in *International Journal of Robotic Research*, vol. 12, no.3, pp. 274-287

Rus D. and Vona M. 1999. Self-reconfiguration Planning with Compressible Unit Modules. In *Proc. IEEE International Conference on Robotics and Automation*, Chi cargo, IL, pp 2513-2520.

Rus D. and Vona M. 2000. A Physical Implementation of the Self-Reconfiguring Crystalline Robot. In *Proc. IEEE International Conference on Robotics and Automation*, San Francisco, CA.pp. 1726-1733

Unsal C. and Khosla P. 2000. Mechatronic Design of a Modualr Self-Reconfiguring Robotic System. In *Proc. IEEE International Conference on Robotics and Automation*, San Francisco CA. pp. 1742-1747

Walker I. and Cavallaro J. 1999. Keeping the Analog Genie in the Bottle: A Case for Digital Robots. In *Proc. IEEE International Conference on Robotics and Automation*, Chicargo, IL, pp 1063-1070.

Will P., Castano A. and Shen W-M., 1999. Robot modularity for self-reconfiguration. In *SPIE International Symposium on Intelligent Systems and Advanced Manufacturing Proc Vol. 3839.* pp 236-245

Yim M. 1994, New Locomotion Gaits. In *Proc. IEEE International Conference on Robotics and Automation*, San Diego, CA pp. 2508-2514