

Complex Behaviors from Local Rules in Modular Self-reconfigurable Robots

Jeremy Kubica, Arancha Casal*, Tad Hogg

Xerox Palo Alto Research Center, Palo Alto, CA 94304

Abstract

We demonstrate how simple local rules, inspired by social insects, produce complex dynamic behaviors required for locomotion and navigation in modular self-reconfigurable robots. We show how systems made up of many modules respond dynamically to their environment, such as obstacles during navigation. We present control algorithms tested on simulation experiments of TeleCube, a new modular robot developed at Xerox PARC.

1 Introduction

Modular self-reconfigurable (MSR) robots [2, 8, 3, 7, 6, 4, 5] consist of many simple identical modules, that can attach and detach from one another to change their overall connectivity. This means these systems can dynamically adapt their shape to suit the needs of the task at hand, e.g., for manipulation, locomotion, and the creation of static structures.

From a planning and control viewpoint, modular self-reconfigurable robots pose interesting challenges. Because typical systems consist of very large numbers of modules (hundreds or thousands), centralized control and planning schemes must be abandoned in favor of distributed ones. This observation and the fact that each module is a self-contained unit with its own processing, sensing and actuation means that the large body of work on distributed multi-agent control is particularly relevant.

Social insects, such as ants and termites, can be viewed as powerful problem solving systems with sophisticated collective intelligence[14, 15]. Despite the fact that a single insect has very limited intelligence, the group of insects, as a whole, can achieve remarkably complex behaviors needed to coordinate, build and maintain the colony and its nest[16, 17]. Social in-

sects can provide a fitting computational metaphor for modular self-reconfigurable robots, as both systems share basic characteristics: each module has limited capabilities but the ensemble must be able to achieve complex tasks in a distributed fashion, hence the intelligence must lie primarily in the interactions between modules.

Much of the work published to date on modular robots has concentrated on planning the reconfiguration between two static pre-defined shapes, and has often involved small numbers of modules[8, 3, 5, 6, 7, 9, 10]. Lately attention has been turned to the use of Genetic Algorithms to control MSR robots[11, 12] However, dynamically-adaptable robot structures involving hundreds of modules can also be achieved as an “emergent” result of biologically-inspired local rules[1]. This paper further extends that work, in the context of a new type of modular robot, TeleCube, developed and built at Xerox PARC. Specifically, we explore the use of *simple local* rules to produce control algorithms for a common mobile robot scenario: reconfigure, locomote and navigate through obstacles in the environment to reach a target object.

The assumptions we make are as follows:

- Modules have limited computational capabilities, and run a simple finite-state machine (FSM) to switch among different “modes”. State transitions are driven by the local state, the states of neighboring modules, their locations, and some external sensor information.
- Communication is limited to immediate neighbors, and a limited number of bits are exchanged at each step.

The following section describes the robot platform used in our work. The remainder of this paper presents control algorithms for MSR robots that coordinate their actions locally to achieve emergent, global behaviors.

*Contact information: acasal@parc.xerox.com

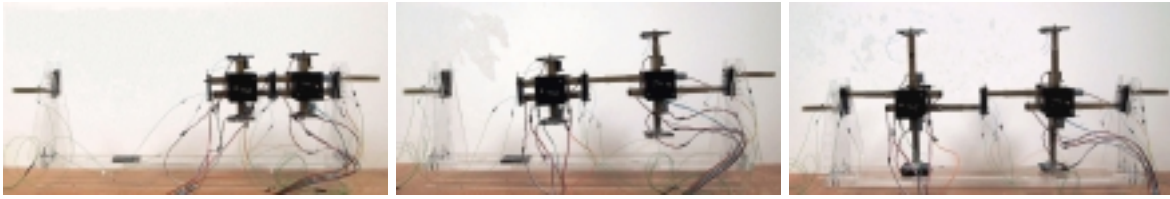


Figure 1: Two TeleCube modules shown with arms retracted and expanded.

2 TeleCubes

2.1 Hardware

TeleCubes is a new MSR robot in which each module is a 3D cube that can prismatically extend each of its six faces independently up to a factor of two times its fully retracted configuration. A similar 2D version was previously developed at Dartmouth [7]. Figure 1 shows a TeleCube module and its telescoping faces. This mechanical design, adopted for its simplicity, has the characteristic of internal motion: modules bound inside a group of modules can move to change their positions. This capability was not possible for a previous MSR, called Proteo[3], in which modules are restricted to motions on the surface of the group. Thus the TeleCube modules allow investigating a different range of behaviors that was possible for Proteo[1].

Other existing modular robot designs include [7, 6, 8, 4, 5, 2]. Even though MSR robots like TeleCube and Proteo require specific attention to their particular motion primitives, the methods presented here are not tied to any specific design, being applicable to MSR robots in general.

2.2 Module Movement

Motions for a TeleCube module are only possible along the directions of the cube's faces. In order to change its position within the robot, a module must "slide" to it, which in general requires breaking and making connections with neighboring modules along any number of its six faces. This is illustrated in Figure 2.

All of the modules use the same motion primitive. The modules first disconnect from all neighbors in directions perpendicular to the direction of motion. Then the actual move is performed by expanding the "back" arm and contracting the "front" arm, where front and back are determined by the direction of motion. Performing the dual expansion and contraction has the advantage of allowing the module to effectively slide forward on its arms, instead of pushing or pulling

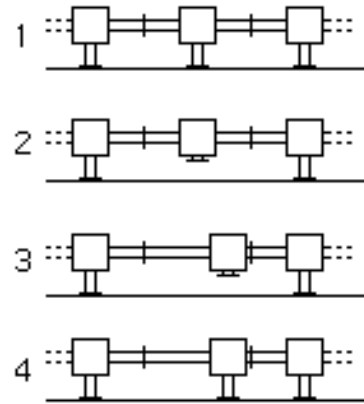


Figure 2: Detail of individual module motion, showing a side view of 3 modules. The middle module moves by contracting its bottom arm (2), then simultaneously expanding and contracting two horizontal arms (3), and finally reextending the bottom arm(4).

large chains of modules. Finally, the module will attempt to connect to any modules to which it is now aligned.

There are two other basic motion primitives, which involve more than one module: dragging modules and cooperative moves. These are illustrated in Figure 3. Dragging can occur when it is possible to pull along a neighbor module that is not connected to anyone else. Cooperative moves are used to help maintain connectedness and are accomplished by two neighbors actively moving at the same time in the same direction without breaking connections.

2.3 Simulator

The experiments were carried out on a simulated version of the TeleCube system. The simulator, written in Java, accounted for the motions of the modules in three dimensions and typically ran with hundreds of modules. Each module moves once per time step,

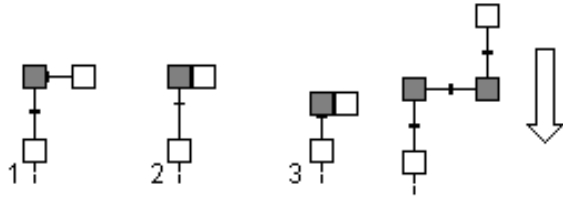


Figure 3: The three steps of a drag move and a cooperative move.

but in order to simulate an asynchronous system the order in which the modules move is random and different each turn. Each of the modules is given a finite strength, allowing the module to move itself and drag up to four other modules, with at most one module per face except for the two faces along the direction of motion.

The simulator makes several simplifying assumptions. First, the module arms can only occupy two states, fully expanded and fully contracted. Second, the modules arms are infinitely rigid, so no drooping occurs when an arm is extended. Both of these assumptions simplify alignment of modules after a move. Lastly, the simulator assumed infinite friction between the module and the floor, effectively eliminating sliding effects.

In addition, a global connectivity check was used to insure disconnecting from a neighbor would not result in separation from the rest of the ensemble. This check may be easily preformed in hardware, and was implemented as a depth-first search of the nodes in the software.

3 Control Approach

Our approach takes advantage of the distributed, homogenous nature of MSR robots. It also scales well as the number of modules increases, whereas conventional centralized schemes become computationally intractable. It has the following characteristics:

There is no central control processor, or designated leader module, or offboard planner. Global motion results, or “emerges”, from purely local control rules executed at every module and through communication between neighboring modules alone.

Each module runs a local FSM control “script”, consisting of simple if-then rules which may act prob-

abilistically. This use of randomization is helpful in preventing modules from becoming indefinitely stuck in unproductive configurations.

We make use of the following concepts[1]:

The **Mode** of a module is its present FSM state, which determines the rules of behavior of a module.

Seeds The **seed** mode is the catalyst for starting new structures and focusing the movement of modules to specific spots.

Scents are the means of global communication among modules. Scents are propagated through the system in a distributed breadth-first fashion [1]. As scents are propagated they act as global gradients to guide motion.

Each module is also capable of communicating with any neighbors in all six directions. This communication falls into three basic categories: messages, queries, and commands. Messages, which include scents and impulses, are information actively sent from one module to another. An impulse is a direct message to a neighbor. A scent, explained above, (which can be either positive or negative) is a message that gets propagated to other modules and whose strength or value changes with each propagation. Queries are requests sent to neighboring modules for basic types of information, including arm length and module state. Finally, commands are forcible messages instructing a neighbor to either contract an arm, extend an arm, or to move in a given direction.

4 Results

We present results for a typical mobile robot task: reach a target object by navigating through an environment with obstacles. MSR robots present the additional requirement of reconfiguring into the appropriate shape as the task progresses.

Figure 4 shows a group of modules reconfiguring into a snake and moving towards a goal location. As it encounters obstacles, it reconfigures to be able to turn and navigate them, finding a narrow passage to squeeze through, again as a snake, and finally reaches the target. From the point of view of control, the task requires three distinct behaviors: Reconfiguration, Locomotion (including turning) and Path Planning (with obstacle avoidance).

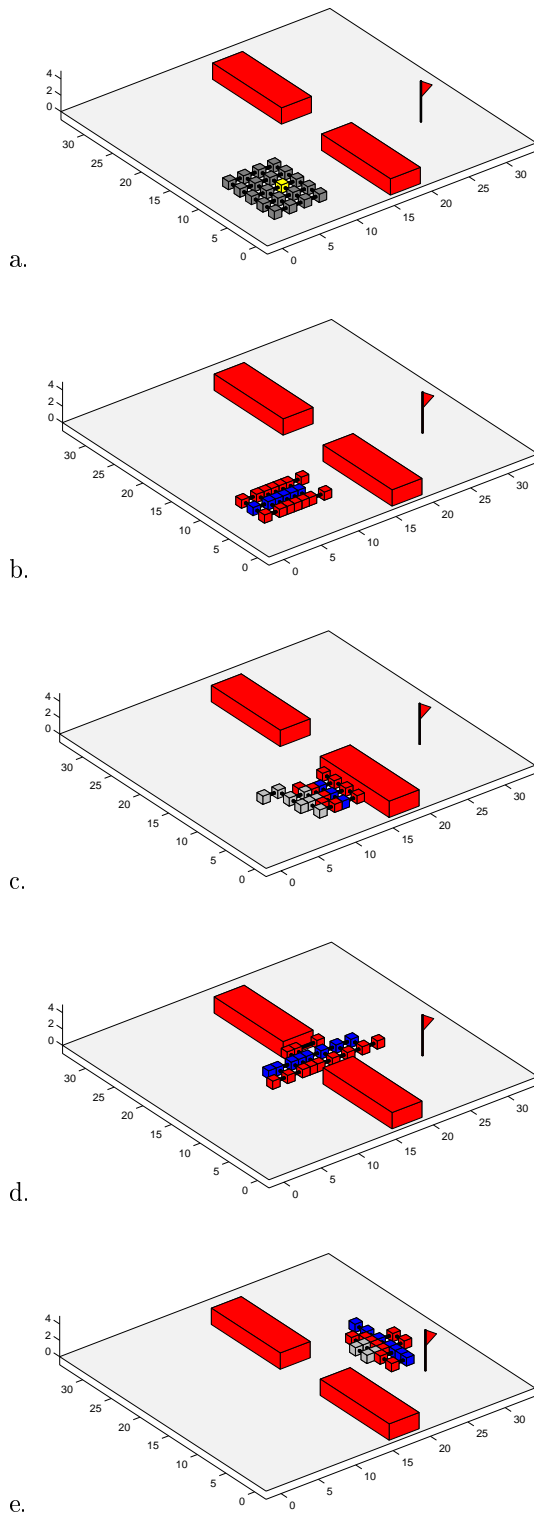


Figure 4: Reconfiguration and movement toward a goal. Module colors represent different modes as described in the text.

The remainder of this paper describes in detail the local control rules required by these behaviors. Although not presented in this paper due to space constraints, this approach has also been used to achieve 3D manipulation of an object fully surrounded by hundreds of modules. This kind of “internal” robot manipulation is a unique capability of the TeleCube module design.

4.1 Modes and Scents

The behaviors are handled via four FSM states or modes (**undefined**, **tissue**, **seed**, and **spine**) and seven scents (assignment, reset, spine-direction, tissue, obstacle, suggested-move and distance-from-spine). In Figure 4, the modules are colored according to their mode: dark gray for **undefined**, light gray for **tissue** modules beyond their distance threshold from the **spine**, red for **tissue** within the distance threshold, blue for **spine** and yellow for **seed**.

A module performs the actions below before any of its current mode-specific actions.

- If an assignment scent is received with a greater priority than the present mode, switch to that mode (mode priorities are, from low to high: **undefined**, **tissue**, **seed**, **spine**).
- If a reset message is received, transition to the **undefined** mode and end turn.
- Propagate a decayed version of any tissue and obstacle scents received to neighbors.
- Propagate spine-direction scents to neighbors.

4.2 Reconfiguration

The Snake configuration is not a set assignment of module positions, but rather a configuration with a given width and height.

The reconfiguration begins when one node is chosen at random to be a **seed** (the yellow module in Figure 4a). This choice does not need to be internal, but could also be triggered by an outside stimulus.

The **seed** module acts as a catalyst to form a “spine”. The spine is a straight line of modules oriented along one of the three global axis, serving as the center of the snake structure. The **seed** module, which immediately turns into a **spine** module, serves as the first module in the spine. The spine then spreads out from the original module in the two directions parallel the spine direction. In addition, the

spine modules emit scents in the perpendicular directions, where receiving modules switch to the **tissue** mode and keep propagating the spine scent gradient. The **tissue** modules try to move in towards the spine as if attracted by a magnetic field. The process of modules following the gradient towards the spine forms a thinner and thinner snake structure and terminates when all of the tissue nodes are within a given threshold from the spine (less than 3 modules wide). Figures 4a and b show the snake being formed from an original square configuration of the modules, but any initial shape could have been used.

The rules for each mode are as follows:

Seed Modules: The module in the **seed** mode catalyses the formation of the spine. Its rules are, in order, as follows:

- If there are no neighbors perpendicular to the direction of the spine but there is a neighbor in the direction opposite the spine direction, send a seed assignment scent to that neighbor, transition to undefined state, and end turn.
- Emit spine assignment scents in directions parallel to the spine direction.
- Emit tissue assignment scents in directions perpendicular to the spine direction.
- Transition to the **spine** mode.

Spine Modules: A module in the **spine** state makes up part of the line of modules forming the spine. A module in this mode proceeds as follows:

- If a suggested-move impulse is received, move in that direction.
- Verify that the spine is not misaligned by checking that no neighbor in a perpendicular spine direction is in the **spine** mode. If there is such a neighbor transition both modules to the **undefined** mode and end the turn.
- Emit spine assignment scents in directions parallel to the spine direction.
- Emit tissue assignment scents in directions perpendicular to the spine direction.
- Maintain uniform density as follows: Check the distances to neighbors along the spine directions by taking the sum of the two arm lengths. If the module is farther from or closer to a neighbor than a distance of one fully extended arm,

then move towards or away from the neighbor respectively with a probability of $1 - ((N_c/8) + .75)$ where N_c is the number of the module's connections perpendicular to the spine. This probability is biased to maintain connectedness by accounting for the number of connections that will be broken and limiting the number of modules moving each turn due to density problems. In the event that a movement is triggered, send a suggested-move message to all neighbors that will be disconnected from before moving, and then move.

Tissue Modules: A module in the **tissue** state simply attempts to get within a threshold distance of the spine. It proceeds as follows:

- If the gradient indicates the module is within the distance threshold from the spine and the distance to the neighbor in the direction towards the **spine** is zero, then with a 10% probability, try to join the spine.
- If the gradient indicates the module is within the distance threshold from the spine, end turn.
- Emit a tissue scent to all neighbors.
- Move toward the spine with a 25% probability, or make a random move with a 10% probability.
- Maintain good density along the spine direction as described in the **spine** mode to maintain uniform density.
- Maintain good density perpendicular to the spine direction as above, but with probability $1 - ((N_c/10) + .475)$. Note that this probability is greater than that for maintaining density along the spine direction.

Experimentally it was found that reconfiguration moved faster and more robustly when there was a large number of density movements perpendicular to the spine direction relative to the number of density movements along the spine direction. The main reason being that parallel movements often left modules without neighbors on which to pull or push their way to the spine.

A **tissue** module's attempt to move closer to the spine can be visualized as an attempt for a module to move down a level, where the levels consist of parallel lines of modules. This move turns out to be both complex and very common in the snake reconfiguration. Therefore the motion was given as a set sequence of actions that includes movements, communications

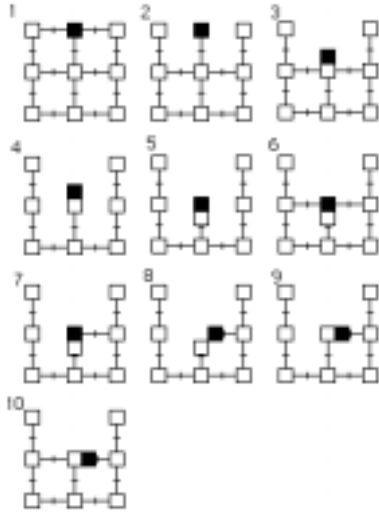


Figure 5: Sequence of actions needed to move down by a row.

with neighbors, and forced moves. The sequence is shown in Figure 5 for the darkened module trying to move into the row below it.

Reconfiguration continues until the tissue scent is no longer detected.

4.3 Locomotion

Once the modules have reached a snake configuration, locomotion of the snake structure begins (Figures 4b-e). Specifically, the locomotion is triggered by the lack of the tissue scent. Thus, when all of the modules have moved within the distance threshold, the scent decays and movement begins. Locomotion along the direction of the spine is accomplished through the use of a movement scent. The foremost module in the direction of motion that is in the spine state acts as the leader for the structure. It initiates the movement by sending all of its neighbors movement scents. When a module receives it, it will move if possible, and in turn propagate the scent to its neighbors. After responding to movement scent, a module will wait a set number of time steps, until its “energy level” has returned to normal, before responding to the next one. The energy cost of a move is set at three, and the module recovers one energy unit per turn. This delay is used to insure the modules move in waves, resulting in a centipede-like gait of expansions and contractions.

Turning is accomplished by changing the location and direction of the spine and reconfiguring into a new



Figure 6: Snake with old spine direction (left), before reconfiguration (center) and after reconfiguration (right).

snake structure around the new spine (Figure 6). A turn begins when any module sends out a reset scent. The reset scent carries with it a direction of the new spine and also information on where the new **seed** module should be relative to the module that released the reset. Once the new **seed** receives the reset scent it transitions to the **spine** mode and begins forming the new spine. All of the other modules transition to the **undefined** state and wait for assignment scents from the spine. Upon completion of the reconfiguration, a snake is formed pointing in the new direction. Because the new spine can only form at angles perpendicular to the old spine, the snake is limited to making right angle turns only.

4.4 Path Planning

In the example of Figure 4, the robot has to navigate its way around two obstacles and the surrounding walls. In order to make it to the goal flag, it has to detect an opening between the obstacles and squeeze past it. We assume the modules have contact sensors to detect when an obstacle is hit. Also, we assume perfect dead-reckoning for localization purposes. To plan a path towards a goal location, the modules track the moves they have made from their original coordinates. In this way, they can determine their global position and relative position to the given goal location.

The modules use local, reactive path planning. At the beginning of the simulation, modules are only given information about their coordinates and the location of the goal module. At the onset of locomotion, the modules are arrayed in a snake configuration along a spine that is oriented in a given direction. The foremost **spine** module in the spine direction becomes the effective “head” of the snake, handling the path planning decisions. This module’s control is limited to sending movement impulses and reset impulses. Movement impulses force the modules in the snake to take a step in the spine direction. Reset impulses cause the

modules to reset and reconfigure along a new spine direction specified in the reset. Thus these two signals can be viewed as step and turn commands.

At its simplest level, path planning consists of repeated attempts to minimize distance to the goal object. If taking a step in the spine direction will serve to decrease this distance then the path planning will take a step in that direction. Otherwise, the snake will make a right-angle turn in the direction that will get it closer to the goal object. When to make a turn is determined by the alignment of the module that is going to be the new **seed** module (the second **spine** module from the front) with the object. Thus moving towards a goal at its simplest involves two directions and a single right-angle turn.

Obstacles in the environment pose additional constraints. When the snake hits an obstacle in the direction of travel it will take a right-angle turn in the direction that will move it closer to the goal. It will then continue in this direction until it hits another obstacle (such as a wall), or aligns itself with the obstacle and waits for an opening it can turn into (see Figure 4c,d). In cases where the goal is directly behind an obstacle, the snake should move away from the obstacle until it can turn around it. Using an obstacle scent the snake tracks where the **tissue** modules on the outside of the spine are touching the obstacle. The snake will only try to turn towards the goal when there is no longer an obstacle in the direction that it wants to turn in. Thus the robot will follow a wall if it hits one.

This approach to dealing with obstacles brings with it the complexity of determining when the snake is truly able to turn. Factors such as the decay rate of the obstacle scent and the contour of the obstacle can lead to undesirable behaviors such as repeatedly trying to turn into the obstacle as the snake moves along its surface and overshooting an opening.

Local path planning has the drawbacks of local minima. A prime example is the case of an L shaped obstacle. If the goal is near the vertex of the L and the snake is inside the L, it will repeatedly turn between two directions to try to minimize the distance to the goal. This can be prevented by adding a timeout, after which the snake performs a random move. Because the same module may not be doing the path planning each time, this knowledge must be shared among all modules in the snake.

5 Conclusions

This paper presents an approach to the control of modular self-reconfigurable robots inspired by social

insects. We show how the use of simple local rules to switch between “modes” allows completion of a complex task, involving reconfiguration, locomotion and navigation in an environment with obstacles.

Complex interactions are produced through the propagation of information and local action guided by simple rules. This approach replaces an emphasis on global planning and centralized control with distributed behavior, local control and communications. We show results in a simulation of TeleCube, a robot developed at Xerox PARC. The control primitives can easily be scaled up in the number of modules, and down in the size of individual modules, and are general enough to fit most modular robot designs.

As the results in this paper and [1] show, the social insect programming metaphor is proving a viable approach to the tremendous growth of complexity in software and control, of which modular robotics is a case in point.

References

- [1] H. Bojinov, A. Casal and T. Hogg, “Emergent Structures in Modular Self-Reconfigurable Robots”, Proc. ICRA2000.
- [2] Yim, M., “Locomotion With A Unit-Modular Reconfigurable Robot”, Stanford University PhD Thesis, 1994.
- [3] Yim, M., Lamping, J, Mao, E., Chase J.G., “Rhombic Dodecahedron Shape for Self-Assembling Robots”, Xerox PARC, SPL TechReport P9710777, 1997.
- [4] Murata S., Kurakawa, H., Kokaji, S., “Self-Assembling Machine”, Proc. of IEEE ICRA’94.
- [5] Murata S. , Kurakawa, H., Yoshida, E., Tomita, K., Kokaji, S., “A 3-D Self-Reconfigurable Structure”, Proc. IEEE ICRA’98.
- [6] Kotay, K., Rus, D., Vona, M., McGray, C., “The Self-reconfiguring Robotic Molecule: Design and Control Algorithms”, Algorithmic Foundations of Robotics, 1998.
- [7] Rus,D., Vona,M., “Self-reconfiguration Planning with Compressible Unit Modules”, Proc. IEEE ICRA’99.
- [8] Pamecha, A., Ebert-Uphoff, I., Chirikjian G.S., “Useful Metrics for Modular Robot Motion Planning”, IEEE

Transactions on Robotics and Automation, Vol.13,
No.4, 1997.

- [9] Casal, A., Yim, M. “Self-Reconfiguration Planning for a Class of Modular Robots”, Proc. SPIE Symposium on Intelligent Systems and Advanced Manufacturing, Sept. 1999, Vol.3839.
- [10] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Kuroda, I. Endo: “Self-Organizing Collective Robots with Morphogenesis in a Vertical Plane”, Proc. IEEE ICRA’98.
- [11] Bennett, F.H., Rieffel, E.G., “Design of Decentralized Controllers for Self-Reconfigurable Modular Robots Using Genetic Programming”.
- [12] Lipson, H. and J. B. Pollack, “Towards Continuously Reconfigurable Self-Designing Robotics” in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, 2000.
- [13] Hackwood, S., Beni G., “Self-organization of Sensors for Swarm Intelligence”, Proc. IEEE ICRA’92.
- [14] M. Resnick, *Turtles, Termites and Traffic Jams, Explorations in Massively Parallel Microworlds*, The MIT Press, 1994.
- [15] E. Bonabeau, M. Dorigo, G. Theraulaz, *Swarm Intelligence, From Natural to Artificial Systems*, Oxford University Press, 1999.
- [16] Johnson, George, “Mindless Creatures Acting ‘Mindfully’ ”, *New York Times* on the Web, March 23, 1999.
- [17] Gordon, Deborah, “Ants At Work: How an Insect Society is Organized ”, W.W. Norton and Company, 2000.