

# A General Constraint-Based Control Framework with Examples in Modular Self-Reconfigurable Robots

Ying Zhang, Markus P.J. Fromherz, Lara S. Crawford and Yi Shang

*Palo Alto Research Center  
3333 Coyote Hill Rd, Palo Alto, CA 94304  
E-mails: {yzhang,fromherz,lrcrawford,yshang}@parc.com*

## Abstract

*In this paper, we advocate a general constraint-based control framework that is highly promising for building control systems with complex dynamic structures, such as modular self-reconfigurable robots. In this framework, a controller consists of constraint solving components distributed in a network of embedded processors. Constraint solvers are goal oriented deliberative agents that can be used as control regulators or as information retrievers. The framework is built on the Attribute/Service Model (ASM), a middleware for coordinating actuators, sensors and tasks in distributed real-time embedded systems. The communications and coordination among the services are realized via shared attributes. Examples of controlling a modular self-reconfigurable robot are illustrated in the paper.*

## 1. Motivation and Introduction

With the increasing computational and communication power in embedded chips, embedded systems with 100s to 100,000s of nodes will become a reality. One example of such systems is modular self-reconfigurable robotic systems, such as PolyBot [17]. PolyBot has been designed for applications including planetary exploration, undersea mining, search and rescue and other tasks in unstructured, unknown environments. PolyBot is composed of a large number of modules that can be disconnected and reconnected automatically in different arrangements to form a new system enabling new functionalities. There are a growing number of modular self-reconfigurable robotic systems [7,10,11,13,15]. These systems claim to have many desirable properties including versatility, robustness and low cost. However, practical applications outside of research have yet to be seen. One outstanding issue for such systems is the increasing complexity for effectively programming a large distributed system, with hundreds or even thousands of

nodes in changing configurations. Designing control systems for such large embedded distributed networks imposes a new challenge.

In this paper, we advocate a general constraint-based control framework that is highly promising for building control systems with complex dynamic structures, such as modular self-reconfigurable robots. In this framework, a controller consists of constraint solving components distributed in a network of embedded processors. Constraint solvers are goal oriented deliberative agents; they can be used as control regulators that drive the system to desired states or as information retrievers that extract current state information from sensed data. The framework is built on the Attribute/Service Model (ASM), a middleware for coordinating actuators, sensors and tasks in distributed real-time embedded systems [20,22]. Here, constraint solvers are services that can be triggered either by clock ticks with a fixed sample time, or by externally (hardware) or internally (software) generated events. The communications and coordination among the services are realized via shared attributes.

Much research has been done in control architectures for distributed robotic systems [1,4]. However, few of these approaches were targeted for large distributed embedded networks. Research in multi-agent systems [14] discusses collaboration among deliberative and reactive agents. However, most of the effort was on software agents, rather than agents embedded in electromechanical systems. Research on Model Predictive Control (MPC) [2,8] applies constrained optimization methods to control systems. However, most of these methods are either for linear systems or so time and memory expensive that they can hardly be applied to embedded systems. Constraint-based control synthesis and sensor computation have also been studied [5,12,18,19,23]. However, most of these were focused on subsystems, and none of these were dealing with systems on this scale and complexity. This paper

moves further in that direction, by building a framework for a concrete, complete and complex embedded distributed system.

This paper is organized as follows. Section 2 presents the constraint-based control framework, defining the functionality of general components such as control regulators and information retrievers, built on the top of the Attribute/Service Model. Section 3 applies the constraint-based control framework to the modular self-reconfigurable robots, with actual components for locomotion and reconfiguration. Section 4 concludes the paper.

## 2. Constraint-Based Control Framework

A general distributed control system consists of two types of components: *control regulators* and *information retrievers*. A control regulator produces desired control efforts to move the system to desired behaviors, while an information retriever estimates the internal or external states from raw sensor data. In most traditional systems, a control regulator is a feedback controller or a simple input/output state automaton; or it can be an arbiter that combines or selects control outputs from various sources. A more complex example in robotics would be the closed-form controller solving the kinematics for the 6 DOF of a typical robotic manipulator. Similarly, an information retriever can be a sensing aggregation that computes mean and derivations from the sensor data; or it can be a pattern recognizer that characterizes the sensor data.

However, many of these traditional approaches do not scale to larger, distributed and dynamically reconfigurable systems. In such systems, control objectives and constraints as well as entire configurations change on-line. At the same time, the redundancy of such large-scale systems offers new opportunities in functionality and fault tolerance. For example, a manipulator may not only be asked to achieve typical goals such as following a trajectory with its end effector, but also to minimize velocities, distribute torques equally, and avoid obstacles. In order to realize the potential of such systems, robust, versatile and scalable control algorithms must be developed. Specifically, we cast many of the control problems in such systems as constrained optimization problems, and we propose a constraint-based control framework that consists of constraint solving components distributed over a large embedded network. In this framework, a constraint solving component is a goal-oriented deliberative agent that can be used as a control regulator or as an information retriever. Constraint solvers can be considered as services, which communicate and

are coordinated through information shared in a generic attribute/service model.

In the rest of this section, we will briefly present the Attribute/Service Model that serves as the communication infrastructure of this approach, and then discuss constraint solving components and the overall structure of the control system in more detail.

### 2.1 Attribute/Service Model

The Attribute/Service Model (ASM) [20,22] provides a general middleware for building applications with multiple threads on multiple processors. ASM has two types of components: *attributes* and *services*. Attributes are abstractions for resources shared among multiple threads located in one or multiple processors; services are abstractions of hardware or software routines.

All attributes are associated with `get()` and `set()` methods. Structures built into the attributes protect the shared resources as well as support synchronization between multiple services. In particular, each attribute is associated with a block of data that has multi-thread protection, i.e., at any given time, there is only one thread that can access the data through the `get()` or `set()` methods. An example of an attribute is a desired joint angle, which may be set by a high level task such as an inverse kinematics service, and used by a motor control service.

All services are associated with `start()`, `stop()`, `suspend()`, `resume()` methods. In general, a hardware service corresponds to setting registers controlling hardware devices. A software service is a thread that is programmed for a particular behavior. An example of a hardware service is actuating a latch for docking; a software service is tracking a desired setting, or solving a set of constraints over time. Services can be event-driven. An event can be associated with many triggers, such as clock ticks, sensor data changes, error discovery or execution completion. For example, a joint controller will be triggered by clock ticks with a fixed sampling time, while a path planner would be triggered by error or completion signals.

All attributes and services are accessible both locally and remotely, where remote is defined as accesses requiring inter-processor communication. Local and remote attributes and services have the same interface so that services at different processors can be coordinated using this middleware infrastructure. ASM has been implemented on vxWorks for Motorola PowerPC 555 with Controller Area Network for communication [20].

## 2.2 Constraint Solvers

In the constraint-based control framework, a constraint solver is associated with a *goal*. A goal is a tuple <input variables, output variables, objective, constraints, time>, where

- Variable: a domain of interest, such as position, velocity or sensor reading. Values of input variables are given to the goal, and values of output variables are solutions to be found.
- Objective: a function of input/output variables
- Constraint: a relation over input/output variables
- Time: the time the goal is to be satisfied, i.e., the values of the input and output variables minimizes the objective while satisfying all constraints.

In our framework, a constraint solver can be *complete* or *partial*. A complete constraint solver finds a solution to the goal; a partial constraint solver finds a step towards the solution. Partial constraint solvers play important roles in embedded distributed control systems, where the cost of communication and computation is often large, and there is a tight closed loop with the dynamics of the system. A constraint solver can be *closed-form* or based on *iterative* methods of numerical computation. The goal and constraints can be given *explicitly* or *implicitly*. The type of solvers to use depends on the dynamics of the problems and the computation resources.

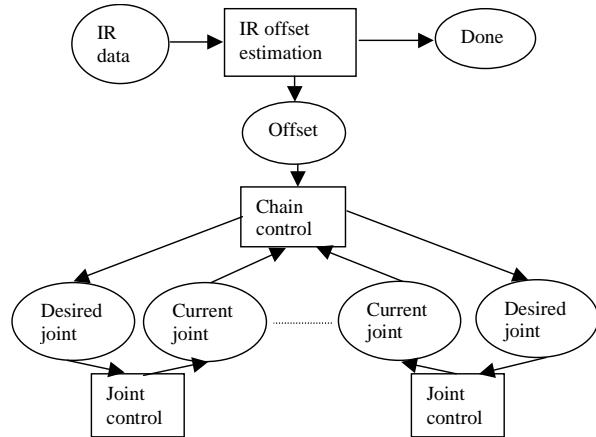
As already indicated, a constraint solver can be a state retriever or a control regulator. For state retrievers, input variables are sensor readings and output variables are state information, where sensor readings are functions of the current state information that is given as a model. The goal is to find the current state information that best fits the sensor readings with respect to the model. A state retriever acts as a diagnostic engine, if the states represent various failure modes. For control regulators, the constraint solving has to close the loop with the plant dynamics. We define a *control variable* to be a variable with a desired value, a current value and a range of valid values. A typical regulator, given the desired value of a control variable, finds the desired values of the control variables of its sub-systems within the valid range; the relationship between the control variable of the system and those of its subsystems is given as a model. More generally, a regulator can maintain certain relationships between the current values of the control variables and the current states of the environment, where the relationship can be represented as an objective function with a set of constraints.

For both state retrievers and control regulators, an error will be generated when the goal cannot be

achieved, and a completion signal will be generated when the goal has been achieved. The events generated will trigger other constraint solvers to modify or re-solve the problem. For example, a tracking service would trigger the path planner if the target is lost or if the target is approached.

## 2.3 Constraint-Based Control Systems

As explained, the constraint-based control system is composed of control regulators and information retrievers, both of which can be constraint solvers. Constraint solvers are provided as services, and input and output variables of the constraint solvers are service attributes shared between them. A control system can be organized in vertical or horizontal layers. In a vertical layer structure, the higher layers are more global computation with less frequent services and lower layers are more local computation with more frequent services. Local sensing information is aggregated and passed upwards and global goal states are decomposed and passed downwards. In a horizontal layer structure, goals can be ordered with priorities, and arbiters based on subsumption architecture can be used to select and combine outputs from constraint solvers. For complex systems, both hierarchies may co-exist. For a large embedded network, components in different layers may be physically located in one processor, and components in the same layer may be physically located at different processors. ASM makes this distinction transparent.



**Figure 1.** Control for docking, where services and attributes are represented as rectangles and ovals, respectively

Figure 1 shows a control subsystem of PolyBot for docking two face plates as a step in reconfiguration, where the IR ranging component estimates the offset between the two plates and the inverse kinematics component computes the desired joint angle.

### 3. Constraint-Based Control for Modular Self-Reconfigurable Robots

PolyBot [17] is a modular reconfigurable robot system composed of two types of modules, one called a *segment* and the other called a *node*. The segment module has two connection plates and one degree of freedom (DOF) of motion. The node module is a rigid cube with six connection plates but no internal DOF. Each module has a Motorola PowerPC MPC555 embedded processor with 448K internal flash ROM and 1M of external RAM. Each module communicates over a global CAN bus with up to 1M bps. A high level CAN protocol Massively Distributed Control Nets (MDCN) has been implemented [21] and ASM has been implemented on the top of MDCN [20,22]. Each connection plate has four IR photo transistors and four IR LEDs for offset ranging, as well as force sensors and latches for docking. Furthermore, each module has two accelerometers for orientation computation with respect to gravity.

#### 3.1 Reconfiguration

Reconfiguration is one of the challenging problems in modular self-reconfigurable robots. The problem of reconfiguration can be decomposed into two steps. The first is a reconfiguration sequence generation, which produces the order of attachments and detachments of the connection plates; the second is docking, which is the process of connecting two plates. There are two important elements in docking: one is offset estimation, which estimates the six degree of freedom offset between the two docking plates, and the other is inverse kinematics, which calculates the desired joint angles and passes them to the joint controllers. The control system for the docking process is illustrated in Figure 2.

Theoretically, the problem of 6 DOF offset estimation is a constraint satisfaction or optimization problem [12,23], i.e., solving  $\langle x, y, z, \alpha, \beta, \gamma \rangle$  given sixteen receiver data readings, one for each pair of emitter and receiver, based on the IR intensity model and emitter-detector positions on the connection plate. A least square minimization can be used for fitting the data. In particular, let  $I_i, i=1..16$ , be the sixteen readings and let  $E$  be an energy function to be minimized, 
$$E = \frac{1}{2} \sum_{i=1}^{16} (I_i - f_i(x, y, z, \alpha, \beta, \gamma))^2$$
 where  $f$  is the model which predicts the IR reading.

IR offset estimation is a state retriever that produces the target 6D position with respect to the current endpoint position of a chain. In order to move the endpoint to the target position, the corresponding movement of joint angles has to be determined. Given a

chain of  $n$  modules, the endpoint is a function of joint angles of the modules in the chain:  $p = f(\theta_1, \theta_2, \dots, \theta_n)$  where  $p$  is  $\langle x, y, z, \alpha, \beta, \gamma \rangle$  and  $\theta_i$  is the joint angle of module  $i$ . Given the desired  $p$ , finding  $\theta_i$  is a constraint satisfaction problem.

In the case of offset estimation, the problem is over-constrained since there are sixteen equations with only six variables. In the case of inverse kinematics, the problem is under-constrained if there are more than six modules in the chain. A chain with a large number of modules is flexible and redundant; in general a closed-form solution does not exist. A general constraint solver is implemented on MPC555 and used for both the offset estimation and inverse kinematics problems. The solver uses Newton's method with Singular Value Decomposition (SVD) for solving linear equations at each Newton step. The use of SVD greatly reduces the risk of reaching a singularity (a common problem when inverting matrices). It also achieves a better result in both under- (minimum change) and over-constrained (minimum error) situations. With a general solver, the system is easy to adapt to changes in the configurations of the chain, the additional objectives and constraints, such as joint angle ranges and torques etc. [5].

It should be noted that constraint-based control encompasses both optimal control, which tries to optimize for an entire known trajectory, and greedy control [8], solving for the next step only. Furthermore, for most closed-loop control, it is not even necessary to obtain the complete solution at every control step. Using the property of iterative methods in most of numerical computations, it often suffices to get a partial solution pointing to the right direction, as long as a stop condition can be detected, i.e., constraints are satisfied. The trade-off between computation resources and quality of solutions depends on the actual task. This property makes constraint-based control distinct from general numerical methods. Research on exploring the actual trade-offs has been conducted [6].

In both the offset estimation and inverse kinematics problems, explicit models of the systems are used. However, even a simple PD controller can be considered as constraint-based [18]; in this case, the model of the plant is implicitly represented in the coefficients of the PD controller. A constraint-based control can also generate non-trivial nonlinear control laws for plants with non-holonomic constraints such as cars [19], and stabilizing head movement in snake-like locomotion, which will be discussed in the next section.

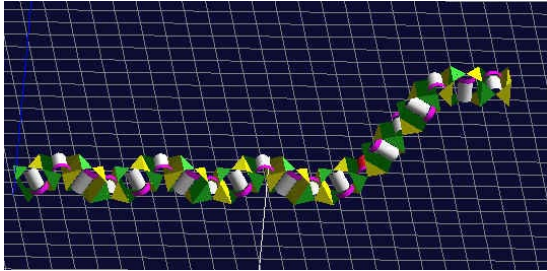
### 3.2 Locomotion

Unlike regular robots, modular reconfigurable robots can have various configurations [16]. Locomotion gaits for most configurations, however, follow the gaits seen in the biological literature on central pattern generators (CPG), where motor neurons between segments or legs introduce a constant phase delay in a traveling wave propagating down the body of the animal [9].

#### 3.2.1 Snake Gaits

Consider a set of PolyBot segments connected linearly as a snake configuration (Figure 2). The segments with rotational axes parallel to the ground are forward motion segments; the segments with rotational axes parallel to the gravity are turning segments.

The general snake gait can be represented by:  $\theta_i(t) = A\sin(\omega t + i\phi) + C$ , where  $\theta_i$  is the joint angle of the  $i$ 'th forward motion segment. In this gait,  $A$ ,  $C$ ,  $\omega$ ,  $\phi$  are attributes that can be changed during locomotion, which in turn, change the speed, the height of obstacles that the snake can traverse, etc.



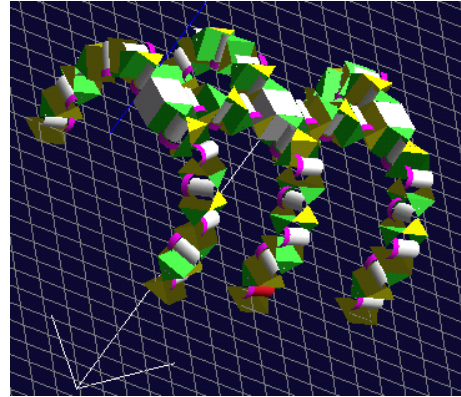
**Figure 2:** Snake motion with head up

In order to hold the head in a straight-level pose for holding a camera, while the rest of the snake body is running snake gait, two specialized modules are configured, one “neck” and one “head”. Each module runs a closed-loop control to regulate the desired orientation of its plate, i.e., the projection of the normal axis of the plate to the gravity vector, for example, 0.8 for neck, 0 for head, such as shown in Figure 3. For each segment module, two accelerometers, each of which has two orthogonal axes, are mounted, using which, the orientation of the module w.r.t. gravity can be estimated [23]. The orientation estimation is a state retriever, even though in this case, a closed form solution is obtained from four equations, corresponding to four readings:  $d_0 = -s\beta$ ,  $d_1 = c\beta s\gamma$ ,  $d_2 = s\beta s\theta + c\beta s\gamma c\theta$ ,  $d_3 = c\beta c\gamma$  where  $s$  stands for sine and  $c$  stands for cosine and  $\theta$  is the joint angle of the module. Given a module orientation  $\langle \alpha, \beta, \gamma \rangle$ , the projection of the normal axis

of its plate to the reference Z-axis (gravity) can be calculated as  $f(\theta) = -s\beta s\theta + c\beta s\gamma c\theta = d_0 s\theta + d_1 c\theta$ . Given  $d$  as desired projection, the constraint-based controller is to solve constraint  $d = f(\theta)$ , or to minimize  $(d - f(\theta))^2$ . The control law, using the gradient method,  $\Delta\theta = (d - f(\theta)) * f'(\theta)$ , i.e.,  $\Delta\theta = (d - d_0 s\theta - d_1 c\theta) * (-d_0 c\theta - d_1 s\theta)$ , is derived, where  $\theta$  can be obtained from the joint sensor. Using this control law for both the neck and head modules, with  $d$  set to 0.8 for the neck and  $d$  set to 0 for the head, the snake is able to move forward while keeping its head straight for holding a camera. The result is validated with the PolyBot simulation.

#### 3.2.2 Centipede Gaits

Centipedes are configurations with  $2 \times n$  legs, as in Figure 3. The centipede gaits we develop, similar to snake gaits, follow a periodic pattern, with phase shift between legs. More complicated gaits for complex terrains have explored constraint-based optimization [3] for foot placement, which fits in to our framework as well.



**Figure 3:** 6-leg centipede configuration

Each leg controller runs a finite state automaton, with six states: **up**, **swing**, **down**, **hold**, **reset** and **rest**. The time duration of each state and the height and width of the step are attributes of the controller, which can be changed during locomotion. In order to achieve high performance, instead of a general constraint solver, a closed-form solution of the inverse kinematics for three active joints is used. Given the height and width of a step, the joint angles of the three active joints are computed to control the up, swing and down motion of the leg. The body control, on a higher level, determines the height and width of a step, the type of gait (wave or equal phase), and the duty cycle of the supporting phase, given the type of terrains. It can also reconfigure the legs by selecting different active joints so that they are best suited to the environment. The body also takes charge of speed and direction change. The centipede can not only

turn, by letting the step size of the left and right legs be different, but also walk “sideways”, by changing the directions of the steps.

#### 4. Conclusion

We have presented a general constraint-based control framework for building complex distributed control systems such as modular self-reconfigurable robots. In this framework, a controller consists of constraint solving components distributed in a network of embedded processors. A constraint solving component acts as a control regulator that outputs desired state at every step, or as a state retriever that given raw sensor data, estimates the current state information. This framework has been successfully applied to modular self-reconfigurable robots, a complex, large, distributed, embedded network system.

#### Acknowledgements:

This work is funded in part by the Defense Advanced Research Project Agency (DARPA) contract # MDA972-98-C-0009 and contract # F33615-01-C-1904.

#### References

- [1] J.S. Albus, “4-D/RCS Reference Model Architecture for Unmanned Ground Vehicles,” *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 3260-3265, 2000.
- [2] A. Bemporad, M. Morari, V. Dua, E. N. Pistikopoulos, “The Explicit Solution of Model Predictive Control via Multiparametric Quadratic Programming,” *Proc. of American Control Conference*, Chicago, Illinois, June 2000
- [3] C.H. Chen, V. Kumar, Y. Luo, “Motion Planning of Walking Robots Using Ordinal Optimization”, *IEEE Robotics & Automation Magazine*, June, 1998
- [4] E. Coste-Maniere, R. Simmons, “Architecture, the Backbone of Robotic Systems,” *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 67-72, 2000.
- [5] M. Fromherz, W. Jackson, “Predictable Motion of Hyper-redundant Manipulators Using Constrained Optimization Control,” *Int. Conf. on AI 2000*, Las Vegas, NV, June 2000, pp. 1141-1148.
- [6] M. Fromherz, L. Crawford, C. Guettier, Yi Shang, “Distributed Adaptive Constrained Optimization for Smart Matter Systems”, *AAAI Spring Symposium for Intelligent Distributed and Embedded Systems*, 2002
- [7] K. Kotay, D. Rus, M. Vona, C. McGray, “The Self-reconfiguring Robotic Molecule,” *Proc. of the IEEE International Conf. on Robotics and Automation*, pp424-431, May 1998.
- [8] E. Lavretsky, “Greedy Optimal Control”, *Proc. of American Control Conference*, Chicago, Illinois, June 2000
- [9] K.A. McIsaac, J.P. Ostrowski, “A Geometric Approach to Anquilliform Locomotion: Modelling of an Underwater Eel Robot”, *Proc. of the IEEE International Conf. on Robotics and Automation*, Detroit, Michigan, May 1999.
- [10] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, S. Kokaji, “A 3D Self-Reconfigurable Structure,” *Proc. of the IEEE International Conf. on Robotics and Automation*, pp432-439, May 1998.
- [11] A. Pamecha, C. Chiang, D. Stein, G.S. Chirikjian, “Design and Implementation of Metamorphic Robots,” *Proc. of the 1996 ASME Design Engineering Technical Conf. and Computers in Engineering Conf.*, Irvine, California, August 1996.
- [12] K. Roufas, Y. Zhang, D. Duff, M. Yim, “Six Degree of Freedom Sensing for Docking using IR LED Emitters and Receivers,” *Experimental Robotics VII, Lecture Notes in Control and Information Science*, ed. D. Rus and S. Singh, Springer, p. 91-100, 2001
- [13] C. Unsal, P.K. Khosla, “Solutions for 3-D Self-reconfiguration in a Modular Robotic System: Implementation and Path Planning,” *Proc. of SPIE Sensor Fusion and Decentralized Control in Robotic Systems III*, Vol. 4196.
- [14] G. Weiss, ed. “Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence,” MIT Press, 1999.
- [15] P. Will, A. Castano, W-M Shen, “Robot modularity for self-reconfiguration,” *SPIE Intl. Symposium on Intelligent Sys. and Advanced Manufacturing, Proceeding Vol. 3839*, pp. 236-245, Sept. 1999.
- [16] M. Yim, “New Locomotion Gaits,” *Proc. of the IEEE International Conf. on Robotics and Automation*, pp. 2508-2514, May 1994.
- [17] M. Yim, D. Duff, K. Roufas, “PolyBot: a Modular Reconfigurable Robot” *Proc. of the IEEE Int. Conf. on Robotics and Automation*, April 2000.
- [18] Y. Zhang, “A Foundation for the Design and Analysis of Robotic Systems and Behaviors”, PhD Thesis, University of British Columbia, 1994.
- [19] Y. Zhang, A.K. Mackworth, “Synthesis of Hybrid Constraint-Based Controllers”, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pp552-567, 1995.
- [20] Y. Zhang, K. Roufas, M. Yim, “Software Architecture for Modular Self-Reconfigurable Robots”, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Hawaii, 2001.
- [21] Y. Zhang, K. Roufas, M. Yim, “Massively Distributed Control Nets for Modular Self-Reconfigurable Robots”, *AAAI Spring Symposium for Intelligent Distributed and Embedded Systems*, 2002
- [22] Y. Zhang, M. Yim, K. Roufas, C. Eldershaw, “Attribute/Service Model: Design Patterns for Distributed Coordination of Sensors, Actuators and Tasks”, submitted to *Workshop on Embedded Systems Codesign*, 2002.
- [23] Y. Zhang, K. Roufas, C. Eldershaw, M. Yim, D. Duff, “Sensor Computations in Modular Self-Reconfigurable Robots”, *8th International Symposium in Experimental Robotics*, July 8-11, 2002