# Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots

Wei-Min Shen, *Member, IEEE*, Behnam Salemi, and Peter Will, *Member, IEEE*

***Abstract*—This paper presents a biologically inspired approach to two basic problems in modular self-reconfigurable robots: adaptive communication in self-reconfigurable and dynamic networks, and distributed collaboration between the physically coupled modules to accomplish global effects such as locomotion and reconfiguration. Inspired by the biological concept of hormone, the paper develops the adaptive communication (AC) protocol that enables modules continuously to discover changes in their local topology, and the adaptive distributed control (ADC) protocol that allows modules to use hormone-like messages in collaborating their actions to accomplish locomotion and self-reconfiguration. These protocols are implemented and evaluated, and experiments in the CONRO self-reconfigurable robot and in a Newtonian simulation environment have shown that the protocols are robust and scaleable when configurations change dynamically and unexpectedly, and they can support online reconfiguration, module-level behavior shifting, and locomotion. The paper also discusses the implication of the hormone-inspired approach for distributed multiple robots and self-reconfigurable systems in general.**

***Index Terms*—Self-reconfigurable robots, self-reconfigurable systems, adaptive communication, dynamic networks, distributed control, multi-agent systems.**

## I. Introduction

SELF-RECONFIGURABLE robots, in one class, are made of autonomous modules that can connect to each other to form different configurations. The connections between modules can be changed autonomously by actions of the modules themselves. Furthermore, since each module is autonomous (has its own controller, communicator, power source, sensors, actuators, and connectors), modules in a self-reconfigurable robot must collaborate and synchronize their actions in order to accomplish desired global effects. Because of this dynamism, solutions must be provided so that communication and control among modules can be adaptive to topological changes in the network.

As an example of a chain-typed self-reconfigurable robot, Fig. 1 shows the CONRO robot system made of small-sized modules that can autonomously and physically connect to each other to form different configurations such as chains, trees, (e.g., legged-bodies), or loops. The top left picture shows a single autonomous CONRO module; the top right picture shows a CONRO chain (snake) configuration with eight modules, the bottom left has two CONRO insects (tree configuration) each of which has six modules for legs and three modules for the spine, and the bottom right is a CONRO loop configuration with eight modules. Each configuration can perform its locomotion, and the robot can autonomously change configurations in limited situations. For movies and more information about CONRO robots, including automatic docking, please visit http://www.isi.edu/conro.

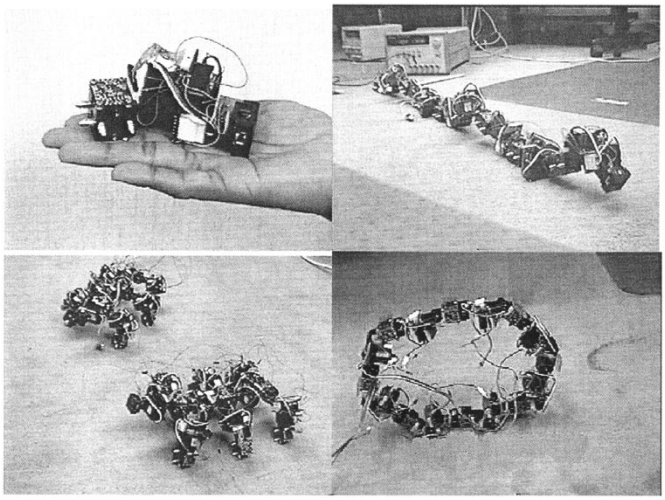


Fig. 1. CONRO module and snake, insects, and rolling track configurations.

This paper addresses two basic problems for modular self-reconfigurable robots: how modules in these robots communicate with each other when connections between them may be changed dynamically and unexpectedly (thus changing their communication routing), and how these physically coupled modules collaborate, in a distributed manner, their local actions to accomplish global effects such as locomotion and reconfiguration. The solutions to these problems may also be applicable to self-reconfigurable systems in general. Examples of such systems include distributed sensor networks [1] and swarm robotic systems [2].

Specifically, modules in a self-reconfigurable robot must coordinate their actions to achieve given missions. Such coordination must be *distributed*, to host a large number of autonomous modules; *dynamic*, to deal with the changes in network topology; *asynchronous*, to compensate the lack of global

Manuscript received April 17, 2002. This paper was recommended for publication by Associate Editor L. Parker and Editor S. Hutchinson upon evaluation of the reviewers' comments. This work was supported by DARPA/MTO under Contract DAAN02-98-C-4032, and by AFOSR under Award F49620-01-1-0020 and Award F49620-01-1-0441.

The authors are with the Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292 USA (e-mail: shen@isi.edu).

Digital Object Identifier 10.1109/TRA.2002.804502

clocks; *scalable*, to support shape-changing and enable global efforts based on weak local actuators; and *reliable*, to recover from local damages in the system and provide fault-tolerance.

In the context of communication, a self-reconfigurable robot can be viewed as a network of nodes that can change and reconfigure their connections dynamically and autonomously. Messages in normal practice are passed between connections using named addresses (such as in the Internet) and are routed from the source to the destination. Various addressing and routing strategies are possible: Single messages can go from one module to the next one; Broadcast messages go to all nodes directly; Multicast messages go to several specific nodes. Routing may be *best-effort* as in the Internet, or *source-routed* as in some supercomputers [3]. Dynamically changing the topology requires continually determining the address and computing the route. This needs continual rediscovery of connection topology at the module level. Each module should discover and monitor unexpected local topology changes in the network, and adapt to such changes by reorganizing its relationships with other modules using their *connectors*. The concept of connector is widely applicable to many different types of networks. For example, in a supercomputing network the connectors are the channels that connect nodes to their neighbors [3]. In a wireless network, the connectors of a node are the channels available for communication. In self-reconfigurable robots, the connectors are physical so that a link is a physical coupling and a network of nodes can form physical structures with different shapes and sizes. For example, the physical connectors in CONRO must be joined and disjoined physically to change shape. Such changes in the network topology make a CONRO robot a dynamic network.

The control of the motion or locomotion of reconfigurable robotics, due to the frequent changes in topology, presents another special challenge since the action messages may need to be directed to the modules doing a specific function rather than to a specific module. Ideally, the modules should coordinate their actions by their locations in the current configuration, not by their names or identifiers. For example, the message should be sent to the "knee" module in the present configuration not to module #37 that perhaps was the knee on the old configuration. With this ability, a module should be able to automatically switch its behavior if its role/location is changed in the configuration. Furthermore, a control message may also require concerted actions. In other words, the message intent may be to execute an action for the robot to "go forward" rather than require the sending of several messages to swing the hip, bend the knee, bend the ankle, and flex the toes and do this in spite of different modules being swapped into and out of the configuration as the system evolves.

This paper presents a biologically inspired approach to address the above challenges and mimic the concept of *hormones* used among biological cells for both communication and control. A biological organizm can have many hormones acting simultaneously and without interfering with each other, each hormone affecting only specific targeted sites. The main idea is that a single "hormone" signal can propagate through the entire network of modules, yet cause different modules to react differently based on their local "receptors," sensors, topology connections, and state information. Computationally speaking, a hormone signal is similar to a content-based message but has the following unique properties: 1) it has no specific destination; 2) it propagates through the network; 3) it may have a lifetime; and 4) it may trigger different actions for different receivers. Notice that hormone propagation is different from message broadcasting. A single hormone may cause multiple effects on the network and different nodes may behave differently when receiving the same hormone. Furthermore, there is no guarantee that every node in the network will receive the same copy of the original signal because a hormone signal may be modified during its propagation.

To apply this idea to adaptive communication, we view each module in a dynamic network as an active cell that can continuously discover its local topological changes and adjust its communication strategy accordingly. We design the adaptive communication (AC) protocol for all modules to discover and monitor their local topology and ensure the correct propagation of hormone messages in the network. This property holds regardless of the changes in the network topology.

To support distributed control with dynamic network topology, we view locomotion as the effect achieved by the interaction on the environment of executing a certain set of actions intrinsically in the robot. For instance, an automobile moves forward when the running engine is engaged with the wheels, provided among other things that there is enough friction between the tires and the road. In our robot we execute a certain set of intrinsic motions and the interaction of these motions with the environment causes locomotion. Motion execution is thus execution of module actions in the robot connection topology plus its interaction with the environment. The hormone concept described above in the context of topology discovery applies equally well to motion execution. We have designed the adaptive distributed control (ADC) protocol for this purpose and applied it to the control of CONRO-like self-reconfigurable robots.

The rest of the paper is organized as follows. Section II discusses the related work. Section III presents a general method for topology discovery and the AC protocol. Section IV extends the AC protocol to the ADC protocol for both distributed control and adaptive communication among self-reconfigurable modules. Section V presents the experimental results of applying the hormone-inspired control protocols to the CONRO robot. Finally, Section VI discusses some fundamental questions about the hormone inspired approaches and suggests future research directions.

## II. Related Work

The communication and control of self-reconfigurable robots is a challenging problem and the approaches for the problem can be either centralized or distributed. From the viewpoint of flexibility and reliability, the distributed approaches are generally preferred for the self-reconfigurable robots. Two recent general articles [4], [5] have provided a good survey of the field.

Related work for centralized control includes Yim *et al.* [4], [6] in which configuration-dependent gait control tables are used to specify actions for each module for each step.

Chirikjian *et al.* [7], [8] study the metric properties of reconfigurable robots and Chirikjian and Burdick [9] propose a mathematical model for controlling hyper-redundant robot locomotion. Kotay and Rus [10] propose a control algorithm for controlling molecular robots. Castano *et al.* [11], [12] use a centralized approach for controlling locomotion and discovering network topology. Rus and Vona [13] use the Melt-Grow planner for the Crystalline robot. Unsal *et al.* [14] utilize a centralized planner for bipartite self-reconfigurable modules. Most recently, Yoshida *et al.* [15] and Kamimura *et al.* [16] demonstrate the online reconfiguration (reconfiguration while locomotion) using a centralized method.

Related work for distributed control includes Fukuda and Kawauchi's control method for CEBOT [17], the series of control algorithms proposed by Murata *et al.* [18]–[22] for self-assembly and self-repairing robots, the hormone-based distributed control method proposed by Shen *et al.* [23]–[26], and the role-based control method by Stoy *et al.* [27], [28]. Most recently, several distributed methods have been proposed for lattice-based robots, including a "secent"-based approach by Bojinov *et al.* [29], a goal-ordering based approach by Yim *et al.* [30], a parallel planner by Vassilvitskii *et al.* [31], and an automata-based approach by Butler *et al.* [32].

The distributed control method proposed in this paper is different from the previously proposed distributed control methods in several aspects. First, a module selects actions based on multiple sources of local information, including the local topology, the sensory inputs, the local state variables, and most importantly the received hormone messages. Second, the local topology defined in this paper distinguishes the connectors of a neighboring module and treat different connectors differently. In other words, a module knows not only that its connector $c_x$ has a neighbor, but also the name of the connector to which $c_x$ is connected. This provides more power for topology representation. Third, the method proposed here can deal with both locomotion and reconfiguration using the same unified framework. This has been demonstrated through the ability of *distributed* online reconfiguration on a chain-based real robot. To the best of our knowledge, such a demonstration has not been done before. Fourth, the method described here has wider application scope than the Cartesian lattice, and can support modules that have internal deforming actions such as pitch, yaw, and roll.

The concept of hormone has previously inspired several researchers to build equivalent computational systems. Autonomous decentralized systems (ADS) [33], [34] is perhaps the earliest attempt to use hormone-inspired methodology to build systems that are robust, flexible, and capable of doing on-line repair. In ADS, the Content Code Communication Protocol was developed for autonomous systems to communicate not by "addresses" but by the content of messages. The ADS technology has been applied in a number of industrial problems [35], and has the properties of on-line expansion, on-line maintenance, and fault-tolerance. However, ADS systems have yet been applied to self-reconfiguration. Another similar approach is proposed in [36] where markers are passed in a network to dynamically form sets of nodes for performing parallel operations. Finally, biologically inspired control methods have also been used for robot navigation [37].

## III. Adaptive Communication

As described above, the modules in a self-reconfigurable robot are reconfigured structurally. The physical interpretation of this action is that shape morphing occurs. The connectivity interpretation is that the modules have a new communication network topology. The control implication is that global actions such as locomotion require a re-computation of the local actions to be executed by the individual modules. These local actions depend on the position of the module in the reconfigured structure. To the best of our knowledge, such control approach can support some unique and new capabilities, such as distributed and online *bifurcation*, *unification*, and *behavior-shifting*, which have not been seen before in robotics literature. For example, a moving snake robot with many modules may be bifurcated into pieces, yet each individual piece can continue to behave as an independent snake. Multiple snakes can be concatenated (for unification) while they are running and become a single but longer snake. For behavior-shifting, a tail/spine module in a snake can be disconnected and reconnected to the side of the body, and its behavior will automatically change to a leg (the reverse process is also true). In fault tolerance, if a multiple legged robot loses some legs, the robot can still walk on the remaining legs without changing the control program. All these abilities would not be possible if modules could not cope with the topological changes in the communication network.

In this section, we describe an adaptive communication protocol for dynamic networks such as those used in self-reconfigurable robots. Using this protocol, modules can communicate even if the topology of the network is changing dynamically and unexpectedly. Communication with this protocol will be shown to be robust, flexible, and will allow reconfiguration while the network is in operation. The reconfiguration can either be self-initiated, superimposed by external agents, or in response to sensor interaction with the environment.

### A. Self-Reconfigurable Modules and Networks

To illustrate the concept of adaptive communication in a self-reconfigurable network, we will use the CONRO robot as an example. As shown in Fig. 1, a CONRO robot consists of a set of modular modules that can connect/disconnect to each other to form different robot configuration. The detail of a single module is shown in Fig. 2. Each CONRO module is a generalized-cylinder that is 4.0 inch long and 1.0 inch$^2$ in diameter. Every module is autonomous, self-sufficient, and equipped with a microcontroller, two motors, two batteries, four connectors for joining with other modules, and four pairs of infrared emitter/sensor for communication and docking guidance.

The movements of modules are actuated by two servomotors, which provide the pitch (up and down) rotation called DOF1 and the yaw (left and right) rotation called DOF2. With these two degrees of freedom, a single module can wiggle its body and has a limited ability to move. However, when two or more modules connect to form a structure, they can accomplish many

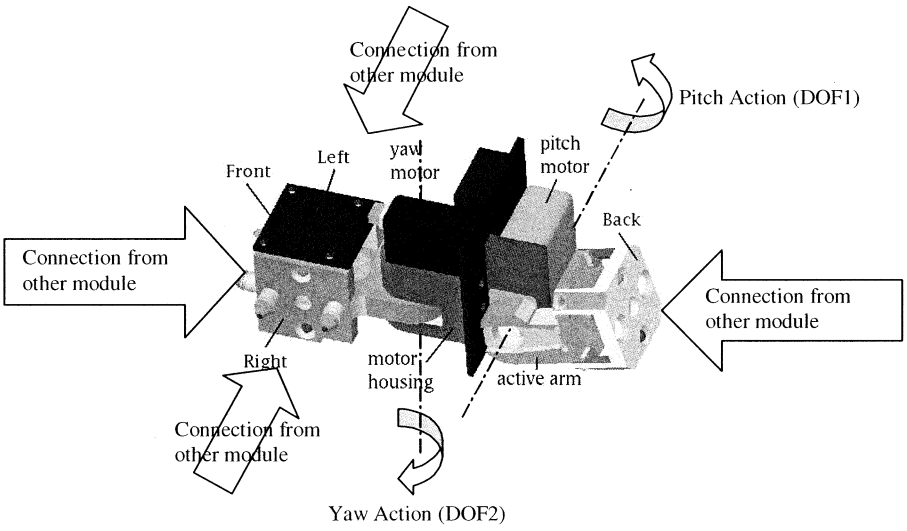



Fig. 2. The schema for a CONRO self-reconfigurable module, and four possible connections to neighbor modules.

different types of locomotion. For example, a chain of modules can mimic a snake or a caterpillar, a body with legs can perform insect or centipede gaits, and a loop can move as a rolling track. Karl Sims [38] has studied this question in details and developed a system for discovering the motion possibilities of different block structures. To some extent, CONRO provides a physical implementation of his results.

The control program on a CONRO module is written in the BASIC language and is running on the on-board STAMP II microcontroller that has only 2 kbytes of ROM for programs and 32 bytes of RAM for variables. Such a tight computational resource poses additional challenges for the control program. We believe that the simplicity and efficiency of hormone-inspired approach has contributed greatly to the successful implementation of all functions and programs on board.

CONRO modules can connect to each other by their docking connectors located at either end of each module. At one end, called the module's *back*, is a female connector, which has two holes for accepting another module's docking pins, a spring-loaded latch for locking the pins, and an shape memory alloy (SMA)-triggered mechanism for releasing the pins. At the other end of a module, three male connectors are located on three sides of the module, called *front*, *left*, and *right*. Each male connector consists of two pins. When a male connector and a female connector are joined together, we call the connection an *active link*. The connected modules are called *neighbors*.

CONRO modules communicate with each other through active links. Each connector has an infrared transmitter and an infrared receiver, and they are arranged in such a way that when two connectors are joined to form an active link, the transmitter and the receiver of one side are aligned with the receiver and the transmitter on the other side, forming a bidirectional local communication link. In CONRO modules, such communication mechanism is established by a handshake between the sender and the receiver. When the sender wants to send a message, it turns on its infrared transmitter and waits for the receiver to respond. When the receiver detects the signal, it will turn on its transmitter and inform the sender and both parties will immediately enter the serial communication protocol (RS232 with 9600 baud rate) and the message will be sent and received. If there is no receiver at the other end, then the sender will not get any response and the procedure will return a timeout failure.

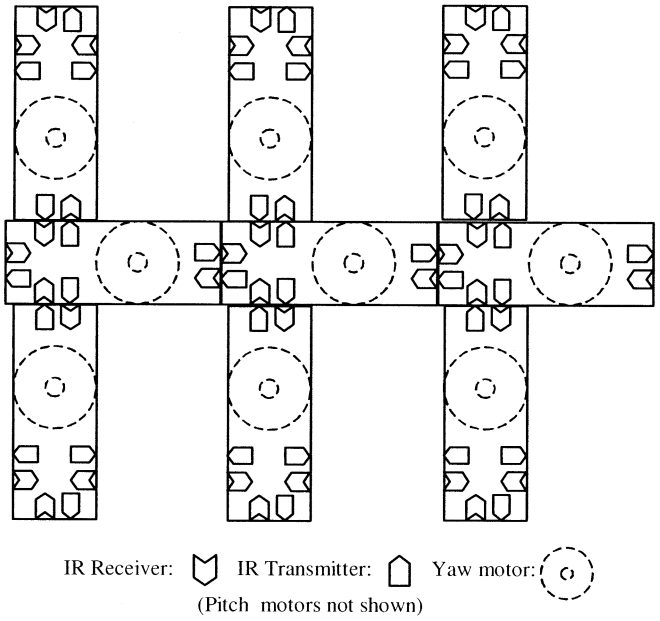



Fig. 3. A top view of a self-reconfigurable communication network among nine CONRO modules.

Fig. 3 shows a network of nine modules (9×4 connectors) forming a hexapod. There are eight active links (which use 16 connectors) and the rest of 20 connectors are still open. Each active link uses two pairs of aligned infrared transmitters and receivers for communication. As we can see from this example, a CONRO robot can be viewed as a communication network of connected modules as well as a physically connected set of modules.

Based on the above description, we define a self-reconfigurable communication network as a connected, undirected graph that has the following properties:

1) each node is a self-reconfigurable module;

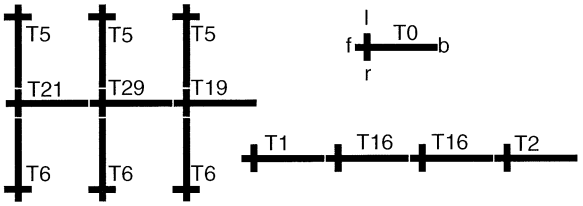



Fig. 4. Some example topological types (T0, T1, T2, T5, T6, T16, T21, T29) of CONRO modules ($f, l, r, b$ are connectors).

2) each node has finite, named connectors. Two connectors of two modules can join and form an active link but one connector can only be in at most one active link.
3) each edge is an active link;
4) the topology of the network may change dynamically, and active links may appear or disappear dynamically;
5) nodes can only communicate through active links;
6) nodes do not know the network size nor have unique IDs.

### B. The Representation of Local Topology

We represent the local topology of a CONRO module in a self-reconfigurable network based on how its connectors are connected to the connectors of its neighbor modules. Shown in Fig. 4, a module is type T0 if it does not connect to any other modules. A module is type T1 if its back connector, $b$, is connected to the front, $f$, of another module. A module is type T2 if its front connector is connected to the back of another module. A module is type T16 if its back is connected to the front of a neighbor and its front is connected to the back of another neighbor. A module is type T21 if its back is connected to the front of another module, and its left, $l$, and right, $r$, are connected to the backs of other two modules, respectively. Note that every active link is a pair of the connector $b$ (the only female connector in a CONRO module) and one of the three male connectors, $f, l$, and $r$. There are 32 types of local topology as listed in Table I and these types are ordered by the number of active links they have. For example, type T0 has no active links; types T1 through T6 have one active link, types T7 through T18 have two active links, types T19 through T28 have three active links, and types T29 through T31 have four active links.

### C. The AC Protocol

Using the concept of hormone messages and local topological types defined above, we can define the AC protocol for continual rediscovery of network topology and ensure adaptive communication. Fig. 5 shows the pseudo-code program for the AC protocol. The main procedure is a loop of receiving and sending (propagating) "probe" hormones between neighbors, and selecting and executing local actions based on these messages. A probe is a special type of hormone that is used for continuously discovering and monitoring local topology. Other types of hormones that can trigger more actions will be introduced later. All modules in the network run the same program, and every module detects changes in its local topology (i.e., the changes in the active links) by sending probe messages to its connectors to discover if the connectors are active or not. The results of this discovery are maintained in the vector variable $\mathrm{LINK}[C]$, where

TABLE I
LOCAL TOPOLOGY TYPES OF CONRO MODULES

TABLE 1: THE LOCAL TOPOLOGY TYPES OF CONRO MODULES

| Connected to other modules: This Module | | | | | | This Module | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $b$ | $f$ | $r$ | $l$ | Type | | $b$ | $f$ | $r$ | $l$ | Type |
| | | | | T0 | | $f$ | $b$ | | | T16 |
| $f$ | | | | T1 | | $f$ | | $b$ | | T17 |
| | $b$ | | | T2 | | $f$ | | | $b$ | T18 |
| | | $b$ | | T3 | | | $b$ | $b$ | $b$ | T19 |
| | | | $b$ | T4 | | $f$ | $b$ | $b$ | | T20 |
| $l$ | | | | T5 | | $f$ | | $b$ | $b$ | T21 |
| $r$ | | | | T6 | | $f$ | $b$ | | $b$ | T22 |
| | $b$ | $b$ | | T7 | | $l$ | $b$ | $b$ | | T23 |
| | | $b$ | $b$ | T8 | | $l$ | | $b$ | $b$ | T24 |
| | $b$ | | $b$ | T9 | | $l$ | $b$ | | $b$ | T25 |
| $l$ | $b$ | | | T10 | | $r$ | $b$ | $b$ | | T26 |
| $l$ | | $b$ | | T11 | | $r$ | | $b$ | $b$ | T27 |
| $l$ | | | $b$ | T12 | | $r$ | $b$ | | $b$ | T28 |
| $r$ | $b$ | | | T13 | | $f$ | $b$ | $b$ | $b$ | T29 |
| $r$ | | $b$ | | T14 | | $l$ | $b$ | $b$ | $b$ | T30 |
| $r$ | | | $b$ | T15 | | $r$ | $b$ | $b$ | $b$ | T31 |

$C$ is the number of connectors for each module (e.g., $C = 4$ for a CONRO module). If there is no active link on a connector $c$ (or an existing active link on $c$ is disconnected), then sending of a probe to $c$ will fail and $\mathrm{LINK}[c]$ will be set to nil. If a new active link is just created through a connector $c$, then sending a probe to $c$ will be successful and $\mathrm{LINK}[c]$ will be updated. After one exchange of probes between two neighbors, both sides will know which connector is involved in the new active link and their LINK variables will be set correctly [1].

The AC protocol has a number of important properties that are essential for adaptive communication in self-reconfigurable networks.

*Proposition 1:* Using the AC protocol, all modules can adapt to the dynamic topological changes in the self-reconfigurable network and discover their local topology in a time less than two cycles of the main loop. The updated local topology information is stored in $\mathrm{LINK}[c]$.

To see this proposition is true, notice that initially all LINK variables have a nil value. If a module has a neighbor on its connector $c$, then $\mathrm{LINK}[c]$ will be set properly when this module receives a probe on that connector. Since every module probes all its connectors in every cycle of the program, the $\mathrm{LINK}[c]$ will be updated correctly with at most two cycles.

*Proposition 2:* If the network is acyclic graph, then the AC protocol guarantees that every nonprobe message will be propagated to every module in the network once and only once. The time for propagating a hormone to the entire network is linear to the radius of the network graph.

To see that Proposition 2 is true, notice that when a new message is generated (e.g., [Test, *, *, *] in Fig. 5), it will be sent to

[1]For example, if an active link is created between the connector $x$ of module A and the connector $y$ of module B, then $\mathrm{LINK}[x] = y$ for module A, and $\mathrm{LINK}[y] = x$ for module B. The $\mathrm{LINK}[C]$ variable represents the local topology type of a CONRO module. For example, a module is type T0 if $\mathrm{LINK}[f, l, r, b] = [\mathrm{nil}, \mathrm{nil}, \mathrm{nil}, \mathrm{nil}]$; type T2 if $\mathrm{LINK}[f, l, r, b] = [b, \mathrm{nil}, \mathrm{nil}, \mathrm{nil}]$; and type T21 if $\mathrm{LINK}[f, l, r, b] = [\mathrm{nil}, b, b, f]$.

```
OUT: the queue of messages to be sent out;
IN: the queue of messages received in the background;
C: the number of connectors for each module;
MaxClock: the max value for the local timer;
LINK[1,…,C]: the status variables for the connectors (i.e., the local topology), and their initially values are nil;
A hormone is a message of [type, data, sc, rc], where sc is the sending connector through which the message is sent, and rc is the
receiving connector through which the message is received.

Main()
LocalTimer = 0;
Loop forever:
  For each connector c=1 to C, insert [probe,_,c,_] in OUT;
  For each received hormone [type, data, sc, rc] in IN, do:
     { LINK[rc] = sc;
          If (type ≠ probe) then
                SelectAndExecuteLocalActions(type, data);
                PropagateHormone(type, data, sc, rc);
     }
  Send();
  LocalTimer = mod(LocalTimer+1, MaxClock);
End Loop.

SelectAndExecuteLocalActions(type, data)
{ // For now, assume that when LocalTimer=0, a module will
  // generate a test hormone to propagate to the network
  // Other possible local actions will be introduced later.
  If LocalTimer==0, then for c=1 to C, do:
     Insert [Test, 0, c, nil] into OUT;
}

PropagateHormone(type, data, sc, rc)
{ For each connector c=1 to C, do:
    If LINK[c]≠0 and c≠rc, then
      { Delete [probe, *, c, *] from OUT;
        Insert [type, data, c, nil] into OUT; // propagation
      }
}

Send()
{ For each connector c=1 to C, do:
     get the first message [type,*,c,*] from OUT,
     Send the message through the connector c;
     If send fails (i.e., time out), LINK[c] = 0.
}
```

Fig. 5. The AC protocol.

all active links from that module. When a module receives a hormone, it will send it to all active links except the link from which the hormone is received. Since the network is acyclic, the generator module can be viewed as the root of a propagation tree, where each module will receive the hormone from its parent, and will send the hormone to all its children. The propagation will terminate at the leaf nodes (modules) where there is no active links to propagate. Since the tree includes every module, the hormone reaches every node. Since every module in the tree has only one parent, the hormone will be received only once by any module.

For networks that contain loops (cyclic graphs), the AC protocol must be extended to prevent a hormone from propagating to the same module again and again. To ensure that each hormone is received once and only once by every module, additional local information (such as local variables) must be used to "break" the loop of communication. We will illustrate the idea in the ADC protocol when we describe the control of rolling tracks, which is a cyclic network.

## IV. Hormone-Inspired Distributed Control

As described above we want a distributed control protocol that is identity free but supports a module to select its actions based on its location in the network. Since hormones can trigger different actions at different site and every module continuously discoveries its local topology, such a control method can be defined based on the hormone messages.

To illustrate the idea, let us first consider an example of how hormones are used to control the locomotion of a metamorphic snake robot. Fig. 6 illustrates a 6-module CONRO snake robot and its caterpillar gait. The types of modules, from the left to the right, in this robot are: T1 (the head), T16, T16, T16, T16, and T2 (the tail). To move forward, each module's pitch motor (DOF1) goes through a series of positions and the synchronized global effect of these local motions is a forward movement of the whole caterpillar (indicated by the arrow). In general, the wavelength of the gait can be flexible (e.g., a single module can craw as a caterpillar). The example in Fig. 6 shows a wavelength of four, but other wavelengths can be defined similarly.

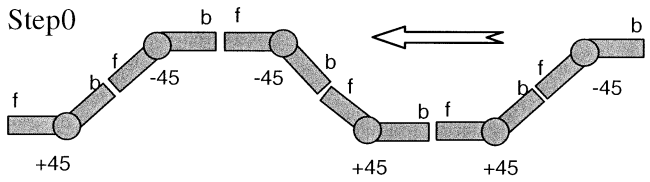



Fig. 6. A caterpillar (or nessie) movement ($b$ and $r$ are connectors, and $+45$ and $-45$ are DOF1).

TABLE II
CONTROL TABLE FOR THE CATERPILLAR MOVE

| Step | Module ID for DOF1 actions | | | | | |
|---|---|---|---|---|---|---|
| | M1 | M2 | M3 | M4 | M5 | M6 |
| 0 | +45° | -45° | -45° | +45° | +45° | -45° |
| 1 | -45° | -45° | +45° | +45° | -45° | -45° |
| 2 | -45° | +45° | +45° | -45° | -45° | +45° |
| 3 | +45° | +45° | -45° | -45° | +45° | +45° |

To completely specify this gait, one can use a conventional gait control table [6] shown in Table II, where each row in the table corresponds to the target DOF1 positions for all modules in the configuration during a step. Each column corresponds to the sequence of desired positions for one DOF1. The control starts out at the first step in the table, and then switches to the next step when all DOF1 have reached their target position in the current step. When the last step in the table is done, the control starts over again at step 0. As we can see in Table II, the six columns correspond to the six module's DOF1 in Fig. 6 (the leftmost is M1, and the rightmost is M6). The first row in this table corresponds to Step 0 in Fig. 6.

The problem of this conventional gait table method is that it is not designed to deal with the dynamic nature of robot configuration. Every time the configuration is changed, no matter how slight the modification is, the control table must be rewritten. For example, if two snakes join together to become one, a new control table must be designed from scratch. A simple concatenation of the existing tables may not be appropriate because their steps may mismatch. Furthermore, when robots are moving on rough ground, actions on each DOF cannot be determined at the outset.

To represent a locomotion gait using the hormone idea, we notice that Table II has a "shifting" pattern among the actions performed by the modules. The action performed by a module $m$ at step $t$ is the action to be performed by the module $(m-1)$ at step $(t+1)$. Thus, instead of maintaining the entire control table, this gait is represented and distributed at each module as a sequence of motor actions $(+45°, -45°, -45°, +45°)$. If a module is performing this caterpillar gait, it must select and execute one of these actions in a way that is synchronized and consistent with its neighbor module. To coordinate the actions among modules, a hormone can be used to propagate through the snake and allow each module to inform its immediate neighbor what action it has selected so the neighbor can select the appropriate action and continue the hormone propagation. This example also illustrates that hormones are different from broadcasting messages because their contents are changing during the propagation.

```
// Built on the AC protocol by adding a RULEBASE and
// extending the following procedure.

SelectAndExecuteLocalActions(type, data)
{ // Select appropriate actions based on
  // type, data, LINK, LocalTimer, and RULEBASE;
  Actions←SelectActions(type,data,LINK,LocalTimer,RULEBASE);
  For each action a in Actions, do ExecuteAction(a);
}

RULEBASE:
{ // The rules here are similar to the receptors in biological cells.
  // They are task-specific "if-then" rules as those in Table 3;
  // Although each desired task has a different set of rules,
  // the rules can be combined together if they are not conflicting.
}
```

Fig. 7. The ADC protocol.

TABLE III
RULEBASE FOR THE CATERPILLAR MOVE

| Module Type | Local Timer | Received Hormone Data | Perform Action | Send Hormone |
|---|---|---|---|---|
| T1 | 0 | | DOF1=+45 | [CP, A, b] |
| T1 | (1/4)*MaxClock | | DOF1=-45 | [CP, B, b] |
| T1 | (1/2)*MaxClock | | DOF1=-45 | [CP, C, b] |
| T1 | (3/4)*MaxClock | | DOF1=+45 | [CP, D, b] |
| T16,T2 | | A | DOF1=-45 | [CP, B, b] |
| T16,T2 | | B | DOF1=-45 | [CP, C, b] |
| T16,T2 | | C | DOF1=+45 | [CP, D, b] |
| T16,T2 | | D | DOF1=+45 | [CP, A, b] |

### A. The Adaptive and Distributed Control Protocol

To implement the hormone-inspired distributed control on the AC protocol, each module must react to the received hormones with appropriate local actions. These actions include the commands to local sensors and actuators, updates of local state variables, as well as modification of existing hormones or generation of new hormones. Modules determine their actions based on the received hormone messages, their local knowledge and information, such as neighborhood topology (module types) or the states of local sensors and actuators.

For these purposes, we specify the ADC protocol listed in Fig. 7. The ADC protocol is the same as the AC protocol except that there is a RULEBASE and the procedure SelectAndExecuteLocalActions( ) is extended to select and execute actions based on the rules in the RULEBASE. The selection process is based on: 1) local topology information (such as LINK[ ] and the module type); 2) the local state information (such as local timer, motor, and sensor states); and 3) the received hormone messages. Biologically speaking, the rules in RULEBASE are analogous to the receptors in biological cells, which determine when and how to react incoming hormones. A module can generate new hormones when triggered by the external stimuli (e.g., the environmental features such as color or sound) or by a received hormone message. When there are multiple active hormones in the system, the modules will negotiate and settle on one hormone activity.

To illustrate the idea of action selection based on rules, let us consider how the caterpillar movement is implemented. The required rules for this global behavior are listed in Table III. In this table, the type of the hormone message is called CP, and

the data field contains the code for DOF1. The other fields of hormones are as usual, but we only show the field of sender connector (sc) for simplicity.

All modules in the robot have the same set of rules, but they react to hormones differently because each module has different local topology and state information. For example, the first four rules will trigger the head module (type T1) to generate and send (through the back connector $b$) four new hormones in every cycle of MaxClock, but will have no effects on other modules. The last four rules will not affect the head module, but will cause all the body modules (T16) to propagate hormones and select actions. These modules will receive hormones through the front connector $f$ and propagate hormones through the back connector $b$. When a hormone reaches the tail module (T2), the propagation will stop because the tail module's back connector is not active. The speed of the caterpillar movement is determined by the value of MaxClock. The smaller the value is, the more frequent new hormones will be generated, thus faster the caterpillar moves.

Compared to the gait control table, the ADC protocol has a number of advantages. First, it supports online reconfiguration and is robust to a class of shape alterations. For example, when a snake is cut into two segments, the two disconnected modules will quickly change their types from T16 to T2, and from T16 to T1, respectively (due to the AC protocol). The new T1 module will serve as the head of the second segment, and the new T2 module will become the tail of the first segment. Both segments will continue move as caterpillar. Similarly, when two or more snakes are concatenated together, all the modules that are connected will become T16, and the new snake will have one head and one tail, and the caterpillar move will continue with the long snake. Other advantages of this hormone-inspired distributed control protocol include the scalability (the control will function regardless of how many modules are in the snake configuration) and the efficiency (the coordination between modules requires only one hormone to propagate from the head to the tail). Let $n$ be the number of modules in the snake, then the ADC protocol requires only $O(n)$ message hops for each caterpillar step, while a centralized approach would require $O(n^2)$ message hops because $n$ messages must be sent to $n$ modules.

In general, the ADC protocol has the following properties.

- *Distributed and Fault-Tolerant:* There are no permanent "brain" modules in the system and any module can dynamically become a leader when the local topology is appropriate. Damage to single modules will not paralyze the entire system.
- *Collaborative Behaviors:* Modules do not require unique ID's yet can determine their behaviors based on their topology types and other local information. The global behaviors can be locomotion or self-reconfiguration.
- *Asynchronous Coordination:* No centralized global real time clocks are needed for module coordination, and actions can be synchronized via hormone propagation.
- *Scalability:* The control mechanism is robust to changes in configuration as modules can be added, deleted, or rearranged in the network.

TABLE IV
RULEBASE FOR A LEGGED WALK

| Module Type | Local Timer | Received Hormone Data | Perform Action | Send Hormone |
|---|---|---|---|---|
| T21, T17, T18 | 0 | | Straight | [LG, A, {l,r,b}] |
| T21, T17, T18 | 0.5*MaxClock | | Straight | [LG, B, {l,r,b}] |
| T29, T19, T26, T28 | | A | Straight | [LG, B, {l,r,b}] |
| T29, T19, T26, T28 | | B | Straight | [LG, A, {l,r,b}] |
| T5 | | A | Swing | |
| T6 | | B | Holding | |

### *B. Other Locomotion Examples of the ADC Protocol*

The ADC protocol can be applied to many different robot configurations. All that is required is to provide the appropriate set of rules to the protocol and have the correct initial configuration in place. For example, Table IV lists the set of rules that will enable a legged robot to walk. In this class of configuration, the module types are similar to those shown in Figs. 3 and 4, where a six-legged robot is shown. In other words, the left leg modules are T6, the right leg modules are T5, the head is T21, the tail T19, and the spine modules are T29. The hormone message used in Table IV is named as LG. We use set notation such as $\{l, b, r\}$ as a shorthand for the set of connectors to send the hormone. The action Straight means $\mathrm{DOF1} = \mathrm{DOF2} = 0$. The action Swing means to lift a leg module, swing the module forward, and then put the module down on the ground. The action Holding means to hold a leg module on the ground while rotating the hip to compensate the swing actions of other legs.

The first two rules indicate that the head module, which can be type T21, T17, or T18, is to generate two new LG hormones with alternative data (A and B) for every cycle of MaxClock. This hormone propagates through the body modules (T29, T26, or T28) and the tail module (T19), alternates its data field, and reaches the leg modules, which will determine their actions based on their types (T5 or T6).

This control mechanism is robust to changes in configurations. For example, one can dynamically add or delete legs from this robot, and the control will be intact. The speed of this gait can be controlled by the value of MaxClock, which determines the frequency of hormone generation from the head module.

As another example of how to use the ADC protocol to control locomotion of self-reconfigurable robots, Fig. 8 shows the configuration of the rolling track. Notice that in this configuration, all modules are of type T16, only their DOF1 values are different. The track moves one direction by shifting the two DOF1 values (90, 90) to the opposite direction.

Table V lists the rules for a rolling track robot. The hormone used here is of type RL, and its data field contains two values of DOF1, and a binary value for selecting the head module. One hormone message continuously propagates in the loop (just as a token traveling in a token ring) and triggers the modules to bend $(\mathrm{DOF1} = 90)$ or straighten $(\mathrm{DOF1} = 0)$ in sequence. We assume that there is one and only one module whose local variable $\mathrm{Head} = 1$. This module is responsible for generating a new hormone when there is no hormone in the loop. This is implemented by the first rule, which will detect a time-out for not receiving any hormone for a long time (i.e., looping through the program for MaxClock times). The head module is not fixed

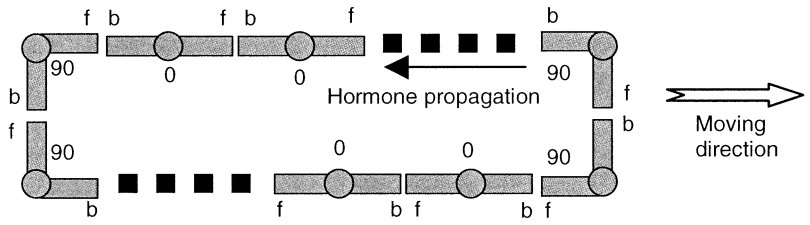



Fig. 8. A rolling track configuration.

TABLE V
RULEBASE FOR A ROLLING TRACK

| Module Type | Local Variables | Received CP Data | Perform Action | Send Hormone |
|---|---|---|---|---|
| T16 | Head=1, Timer=MaxClock | | DOF1=90, Timer=0, Head=0 | [RL,(90,90,1),b] |
| T16 | DOF1=0 | (90,90,1) | DOF1=90, Timer=0, Head=1 | [RL,(90,0,0),b]* |
| T16 | DOF1=0 | (90,90,0) | DOF1=90, Timer=0, Head=0 | [RL,(90,0,0),b] |
| T16 | DOF1=0 | (90,0,0) | DOF1=0, Timer=0, Head=0 | [RL,(0,0,0),b] |
| T16 | DOF1=0 | (0,0,0) | DOF1=0, Timer=0, Head=0 | [RL,(0,0,0),b] |
| T16 | DOF1=90 | (0,0,0) | DOF1=0, Timer=0, Head=0 | [RL,(0,90,0),b] |
| T16 | Head=0, DOF1=90 | (0,90,0) | DOF1=90, Timer=0, Head=0 | [RL,(90,90,0),b] |
| T16 | Head=1, DOF1=90 | (0,90,0) | DOF1=90, Timer=0, Head=0 | [RL,(90,90,1),b] |

Note: * means send the hormone after all local actions are completed.

but moving in the loop. We assume that the initial bending pattern of the loop is correct (i.e., as shown in Fig. 8) and the head module is initially located at the upright corner of the loop. The rules in Table V will shift the bending pattern and the head position in the loop and cause the loop to roll into the opposite direction of hormone propagation. Since hormone propagation is much faster than the actual execution of actions, when a module is becoming the head, it is also responsible for making sure all actions in the loop are completed before the next round starts. The head module will hold the next hormone propagation until all its local actions (DOF1 moving from 0 to 90) are completed.

Notice that the loop configuration is a cyclic network and module types alone are no longer sufficient to determine local actions (in fact all modules in the loop have the same type T16). In general, additional local variables (such as Head) are necessary to ensure the global collaborations between modules in a cyclic network.

Due to the potential of communication errors, there may be situations where no module has the local variable $\mathrm{Head} = 1$ and there is a need for a new head module. In such a case, it may be possible to create a negotiation mechanism for one module to switch its local variable to $\mathrm{Head} = 1$, if there are none in the group—just like some schools of fish where a female changes gender if the male in the group is dead. One possible implementation is to allow any module to self-promote to become a new head if it has not received messages for a long time. In this case, modules must negotiate among to ensure that there is one and only one head in the system. This is sometimes called the problem of *Distributed Task Selection* and we will describe a solution later in Section VI.

### C. Distributed Control of Cascade of Actions

Hormone-inspired distributed control can also be applied to the control of cascade of actions, where actions are organized in a hierarchical structure and a single action in a higher-level can trigger a sequence of lower-level actions. To illustrate the ideas, let us consider the example in Fig. 9, where a CONRO robot is reconfiguring from a quadruped to a snake. The robot first connects its tail with one of the feet, and then disconnects the connected leg from the body so that the leg is "assimilated" into the tail. After this "leg-tail assimilation" action is performed four times, the result is a snake configuration. Note that the middle shape in Fig. 9 is an illustration. In the real CONRO robot, at least 4 modules are needed to make a loop.

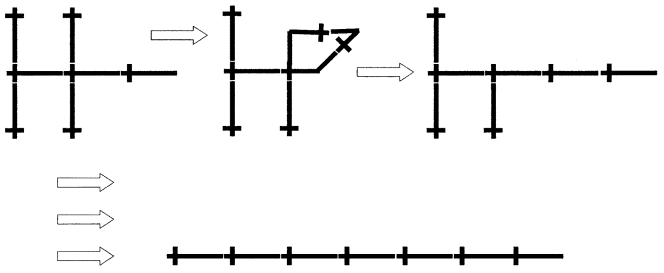


Fig. 9. Reconfiguring a 4-leg robot into a snake body.

To control this reconfiguration, the high-level actions are a sequence of leg-tail assimilations, while the lower-level actions are those that enable the tail to find a foot, to align and dock with the foot, and then disconnect the leg from the body. Using hormones, the control of the reconfiguration can be accomplished as follows. One module in the robot first generates a hormone (called LTS for changing Legs To Snake). This LTS hormone is propagated to all modules, but only the foot modules (which are types T5 or T6) will react. Each foot module will start generating a new hormone RCT to Request to Connect to the Tail. Since there are four legs at this point, four RCT hormones are propagating in the system. Each RCT carries the information about its propagation path [2]. A RCT hormone will trigger the tail module (type T2) to do two things: inhibit its receptor for accepting any other RCT hormones, and acknowledge the sender (using the path information in the received RCT) with a TAR (Tail Accept Request) hormone. Upon receiving the TAR hormone, the selected foot module first terminates its generation of RCT, and then generates a new hormone ALT (Assimilate Leg into Tail) to inform all the modules in the path to perform the lower-actions of bending, aligning, and docking the tail to the foot. The details of these lower-level actions are described elsewhere [26]. When these actions are terminated, the new tail module will activate its receptor for accepting other RCT hormones, and another leg assimilation process will be performed.

[2]A propagation path is a concatenation of all the sender connectors and receiver connectors through which the hormone has been sent so any module along the path can trace back to the original sender.

TABLE VI
HORMONE ACTIVITIES FOR CASCADE ACTIONS

| Hormones | Actions |
|---|---|
| LTS | Start the reconfiguration |
| $RCT_1$, $RCT_2$, $RCT_3$, $RCT_4$ | Legs are activated to generate RCTs |
| TAR, $RCT_2$, $RCT_3$, $RCT_4$ | The tail accepts a RCT, and leg1 stops $RCT_1$ |
| ALT, $RCT_2$, $RCT_3$, $RCT_4$ | The tail and leg1 perform the assimilation process |
| TAR, $RCT_2$, $RCT_4$ | The new tail accepts a RCT, and leg3 stops $RCT_3$ |
| ALT, $RCT_2$, $RCT_4$ | The tail and leg3 perform the assimilation process |
| TAR, $RCT_2$ | The new tail accepts a RCT, and leg4 stops $RCT_4$ |
| ALT, $RCT_2$ | The tail and leg4 perform the assimilation process |
| TAR | The new tail accepts a RCT, and leg2 stops $RCT_2$ |
| ALT | The tail and leg2 perform the assimilation process |
| $\varnothing$ | No more RCT, and end the reconfiguration |

This procedure will be repeated until all legs are assimilated, regardless of how many legs are to be assimilated. In Table VI, we list one possible sequence of hormone activities for assimilating four legs shown in Fig. 9.

## V. EXPERIMENTAL RESULTS

The hormone-inspired adaptive communication and distributed control algorithms described above have been implemented and tested in two sets of experiments. The first is to apply the algorithm to the real CONRO modules for locomotion and reconfiguration. The second is to apply the algorithm to a CONRO-like robot in a Newtonian mechanics simulation environment called Working Model 3D [39].

All modules are loaded with the same program that implements the ADC protocol illustrated in Figs. 5 and 7. For different configurations, we have loaded the different RULEBASE. All modules are running as autonomous systems without any off-line computational resources. For economic reasons, the power of the modules is supplied independently through cables from an off-board power supplier.

For the snake configuration, we have loaded the rules in Table III onto the modules and experimented with caterpillar movement with different lengths ranging from 1 module to ten modules. With no modification of programs, all these configurations can move and snakes with more than three modules can move properly as caterpillar. The average speed of the caterpillar movements is approximately 30 cm/min. To test the ability of on-line reconfiguration, we have dynamically "cut" a ten-module running snake into three segments with lengths of 4, 4, and 2, respectively. All these segments adapt to the new configuration and continue to move as independent caterpillars. We also dynamically connected two or three independent running caterpillars with various lengths into a single and longer caterpillar. The new and longer caterpillar would adapt to the new configuration and continue to move in the caterpillar gait. These experiments show that the ADC protocol is robust to changes in the length of the snake configuration.

To test whether modules can automatically generate hormones when they receive appropriate environmental stimuli from their local external sensors, we have installed two tilt-sensors on one of the modules in the snake configuration, and loaded the following rules to the modules:

If tilt-sensors $= [0, 1]$, generate hormone [FlipLeft, *, *]
If tilt-sensors $= [1, 0]$, generate hormone [FlipRight, *, *]
If tilt-sensors $= [1, 1]$, generate hormone [FlipOver, *, *]

We defined the actions for FlipLeft, FlipRight, and FlipOver for all the modules so that when these hormone messages are received, the modules will perform the correct actions for DOF1 and DOF2 to flip the snake back to its normal orientation. To test this new behavior, we manually pushed the snake, while it is moving as a caterpillar, to its side or flipped it upside down. We observed that the tilt-sensors are activated, new hormones are generated, a sequence of actions is triggered, and the robot flips back to its correct orientation. (See movies at http://www.isi.edu/conro.)

For the legged configuration, we have loaded the rules in Table IV onto the modules and experimented with the various configurations derived from a 6-leg robot (see Fig. 3). These configurations can walk on different number of legs without changing the program and the rules. While a 6-leg robot is walking, we dynamically removed one leg from the robot and the robot can continue walk on the remaining legs. The removed leg can be any of the 6 legs. We then dynamically removed a pair of legs (the front, the middle, and the rear) from the robot, and observed that the robot can continue walk on the remaining 4 legs. We then systematically experimented removing 2, 3, 4, 5, and 6 legs from the robot, and observed that the robot would still walk if the remaining legs can support the body. In other cases, the robot would still attempt to walk on the remaining legs even if it has only one leg. Although we have only experimented robots with up to 6 legs, we believe in general these results can scale up to large configurations such as centipedes that have many legs.

For the rolling track configuration, we have loaded the rules in Table V onto the modules and experimented with rolling tracks with lengths of 8, 10, and 12. In all these configurations, the rolling track moved successfully with speed approximately 60 cm/min. The current configurations must have more than 6 modules and the number of modules must be even. This is because there must be 4 modules with $\mathrm{DOF1} = 90$, and at least two other modules with $\mathrm{DOF1} = 0$. To test the robustness of the system against loss of messages in the communication, we simulated random message losses in the program. We observed that when a message of $[\mathrm{RL}, (*, *, 0), b]$ is lost, the robot will stop rolling momentarily and then the head module's local timer will reach MaxClock, and a new hormone will be generated and the track will resume rolling. If the lost message is $[\mathrm{RL}, (*, *, 1), b]$, then there will be no head module in the system, and the robot will not roll again. However, since most messages are of the first kind, the chance of failing to resume rolling is low. In practice, when message losses do occur, we only observed nonrecovery stops in rare occasions.

In parallel with the experiments on the real CONRO robot, we have also implemented with the ADC protocol on a simulated CONRO-like robot in a software Newtonian simulation environment called Working Model 3D [39]. Using this three-dimensional dynamics simulation program, we have designed a set of virtual CONRO modules to approximate the physical properties of the real modules, including their mass, motor torques, joints, coefficient of friction, moments of inertia, velocities, springs, and dampers. The ADC protocol is implemented in Java and runs on each simulated module. We have experimented with

and demonstrated successful locomotion in various configurations, including snakes with different length (3–12 modules) and insects with different numbers (4–6) of legs. For the cascade actions, we have successfully simulated the reconfiguration sequence described in Section IV.C using the ADC protocol.

## VI. Discussion

This section discusses some related questions about the ADC protocol: (1) How to deal with multiple hormone generators in a robot? (2) How to combine multiple rule sets and switch between them? (3) How to develop the appropriate RULEBASE for a particular global behavior? (4) Is this mechanism applicable to robotic systems in general?

In the above description, all the rules are so designed that one robot has only one hormone generator at a time. For the snake and legged robots, the generator is the head module. For the rolling track, it is the module that has a local variable $\mathrm{head} = 1$. When there are multiple hormone generators in a robot, modules must negotiate to select one and only hormone activity. This problem is sometimes called *Distributed Task Selection*, which is a process for modules to agree and select the same task among multiple initiated tasks in a distributed manner. To solve this problem, we have designed a distributed algorithm called DISTINCT. The main idea is to allow every activated hormone generator to compete to build a spanning tree for itself (being the root of that tree) by propagating a tree-building message to all its neighbors. During this tree building process, if a hormone generator module finds itself being asked to be a part of another tree (when it receives a tree building message from a neighbor module), it will drop its own root status and propagate that message to its neighbors (less the one from which the message is received) and become a part of that tree. If any module that is already in a tree receives another tree building message from a nonparent neighbor module, this module will select one of these received tasks, and designate itself as a new root and start building a new tree by propagating a new tree-building message to all its neighbors. This method is proved to be correct in selecting one and only one hormone activity in a distributed network. Interested readers can refer to [40] for details.

The second question is how to combine multiple rule sets. We notice that as long as rules in both sets do not share the same conditions, then the two different rule sets can be combined into one and the switch between the two behaviors will be automatic. For example, Table III can be combined with Table IV and the result rule set can be used for demonstrating "online behavior-shifting" between caterpillar movement and leg movement. In particular, one can disconnect a tail/spine module from a snake and connect it to the side of the snake, and that module will automatically change its behavior to a leg. A similar but reverse process will change a leg module to a tail/spine module. Using this technique, we can dynamically change a snake configuration to a legged robot by rearranging modules in the body,[3] while the robot is still running.

[3]Currently, this reconfiguration is not yet automatic. We manually disconnect a module from one place and reconnect it to somewhere else in the body. The automatic reconfiguration will be reported in future papers.

The third question is how to develop an appropriate rule set for a particular behavior. We note that the local control rules are similar to the receptors found in biological cells and they determine how modules react to hormones. At the current stage, the development of these rules requires expertise in the expected behavior and the local topological type information about the modules in the configuration. It is still an open problem how to develop these rules automatically. Approaches based genetic algorithms and other machine learning methods can be promising, but further research is needed to generate hormone receptors automatically and correctly. In general, the more complex the behavior is, the more complex the set of rules and requirements are. To make the approach feasible for obtaining for complex behaviors, a general strategy can be suggested based on Simon's hierarchical and nearly-decomposable systems [41]. One first decomposes a complex behavior into a hierarchy of sub-behaviors and design one hormone for each of the most primitive behaviors. Then, another set of hormones is designed to compose the simple behaviors together. The hormones in Table VI are designed using this strategy. As a direction for future research, we will develop software methods to *mechanize* these hormone design procedures.

The fourth question is whether the hormone-inspired approach described here is applicable for robotics systems in general. Although it has been the nature of self-reconfigurable robots that forced us to develop this distributed control mechanism, we believe it would be easily generalized and applied to behavior design of robots for which algorithmic, centralized approaches are usually applied (e.g., wheeled mobile robots). In particular, one can generalize the concept of "connectors between modules" to "communication channels between robots", and then the AC and the ADC protocols can be applied to controlling the collaborations among distributed robotics systems in a dynamic network. Each robot would have a number of "channels" that can be "connected" to other robots' channels to form "active links" which are not necessarily physical couplings but communication links. With this generalization, all the advantages described in this paper could be beneficial to the control of distributed robotics systems. One potential concern for the scalability of the hormone-inspired protocols is that if there are delays in the communication, the system in general may behave erratically. It has been proven that in multi-agent/multi-robot systems, effects of delay may create unforeseen/emerging behavior. As a possible solution for this problem, we propose to use hormones to adjust local timers to compensate the delay. For example, one can image that when a hormone message is received, the module will readjust its local timer as a function of the hormone's lifetime (the number of hops it has been propagated). However, further experiments must be conducted to verify the effectiveness of this proposal.

## VII. Conclusion

This paper presents a hormone inspired control framework for adaptive communication and distributed control in self-reconfigurable robots. The paper argues that self-reconfigurable

robots demand a new methodology for communication and cooperation, and the biological concept of hormone can provide many inspirational ideas. The paper describes the AC protocol for adaptive communication in a dynamic and self-reconfigurable network, and the ADC protocol for distributed control of the actions performed by the nodes in such a network. These protocols are illustrated through the CONRO self-reconfigurable robots. Experiments in both real CONRO modules and in 3-D simulation have shown that this hormone-inspired approach can support many unique features of self-reconfigurable robots, including adaptive communication in dynamic network, decentralized and distributed control for collaboration among autonomous modules, on-line reconfiguration, and scalability to larger and multiple robotic systems.

## ACKNOWLEDGMENT

The authors thank A. Castano, R. Kovac, R. Chokkalingam, and S. Tugsinavisut, for building the CONRO modules, Y. Lu for developing the initial user interface and module communication, and B. Khoshnevis for the initial design of the connectors. They also thank the anonymous reviewers for their insightful comments and suggestions for the earlier drafts of this paper.

## REFERENCES

[1] D. Estrin, R. Govindan, J. S. Heidemann, and S. Kumar, "Next century challenges: Scalable coordination in sensor networks," *Mobile Computing and Networking*, pp. 263–270, 1999.

[2] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. New York: Oxford Univ. Press, 1999.

[3] R. Felderman, A. DeSchon, D. Cohen, and G. Finn, "ATOMIC: A high-speed local communication architecture," *J. High Speed Networks*, vol. 1, no. 1, pp. 1–28, 1994.

[4] M. Yim, Y. Zhang, and D. Duff, "Modular robots," *IEEE Spectrum*, 2002.

[5] D. Rus, Z. Butler, K. Kotay, and M. Vona, "Self-reconfiguring robots," *ACM Commun.*, 2002.

[6] M. Yim, "Locomotion with a Unit-Modular Reconfigurable Robot," Ph.D. dissertation, Dep. Mech. Eng., Stanford Univ., 1994.

[7] G. S. Chirikjian, A. Pamecha, and I. Ebert-Uphoff, "Evaluating efficiency of self-reconfiguration in a class of modular robots," *J. Robot. Syst.*, vol. 13, no. 5, pp. 317–338, 1996.

[8] A. Pamecha, I. Ebert-Uphoff, and G. S. Chirikjian, "Useful metrics for modular robot motion planning," *IEEE Trans. Robot. Automat.*, vol. 13, no. 4, pp. 531–545, 1997.

[9] G. S. Chirikjian and J. W. Burdick, "The kinematics of hyper-redundant robotic locomotion," *IEEE Trans. Robot. and Automat.*, vol. 11, no. 6, pp. 781–793, 1995.

[10] K. Kotay and D. Rus, "Locomotion versatility through self-reconfiguration," *Robot. Autonom. Syst.*, vol. 26, pp. 217–232, 1999.

[11] A. Castano, W.-M. Shen, and P. Will, "CONRO: Toward miniature self-sufficient metamorphic robots," *Autonom. Robots*, vol. 8, pp. 309–324, 2000.

[12] A. Castano and P. Will, "Representing and discovering the configuration of CONRO robots," presented at the *Int. Conf. on Robotics and Automation*, Seoul, Korea, 2001.

[13] D. Rus and M. Vona, "Crystalline robots: Self-reconfiguration with compressible unit modules," *J. Autonom. Robots*, vol. 10, no. 1, pp. 107–124, 2001.

[14] C. Unsal, H. Kiliccote, and P. K. Khosla, "A modular self-reconfigurable bipartite robotic system: Implemenation and motion planning," *Autonom. Robots*, vol. 10, pp. 23–40, 2001.

[15] E. Yoshida, S. Murata, A. Kamimura, K. Tomita, H. Kurokawa, and S. Kokaji, "A motion planning method for a self-reconfigurable modular robot," presented at the *Int. Conf. on Intelligent Robots and Systems*, Maui, HI, 2001.

[16] A. Kamimura, S. Murata, E. Yoshida, H. Kuraokawa, K. Tomita, and S. Kokaji, "Self-reconfigurable modular robot—Experiments on reconfiguration and locomotion," presented at the *Int. Conf. on Intelligent Robots and Systems*, Maui, HI, 2001.

[17] T. Fukuda and Y. Kawauchi, "Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator," *Proc. IEEE Int. Conf. on Robotics Automation*, 1990.

[18] S. Murata, H. Kurokawa, and S. Kokaji, "Self-assembling machine," *Proc. IEEE Int. Conf. on Robotics Automation*, 1994.

[19] E. Yoshida, S. Murata, K. Tomita, H. Kurokawa, and S. Kokaji, "Distributed formation control of a modular mechanical system," presented at the *Int. Conf. on Intelligent Robots and Systems*, 1997.

[20] E. Yoshida, S. Murata, H. Kurokawa, K. Tomita, and S. Kokaji, "A distributed reconifuration method for 3-D homogeneous structure," presented at the *IEEE/RSJ Intl. Conf. Intelligent Robotics and Systems*, 1998.

[21] K. Tomita, S. Murata, H. Kurokawa, E. Toshida, and S. Kokaji, "Self-assembly and self-repair method for a distributed mechanical system," *IEEE Trans. Robot. Automat.*, vol. 15, no. 6, pp. 1035–1045, 1999.

[22] S. Murata, E. Yoshida, H. Kurokawa, K. Tomita, and S. Kokaji, "Self-repairing mechanical systems," *Autonom. Robots*, vol. 10, pp. 7–21, 2001.

[23] W. M. Shen, Y. Lu, and P. Will, "Hormone-based control for self-reconfigurable robots," presented at the *Int. Conference on Autonomous Agents*, Barcelona, Spain, 2000.

[24] W. M. Shen, B. Salemi, and P. Will, "Hormones for self-reconfigurable robots," presented at the Proceedings of the 6th International Conference on Intelligent Autonomous Systems, Venice, Italy, 2000.

[25] B. Salemi, W.-M. Shen, and P. Will, "Hormone-controlled metamorphic robots," presented at the Int. Conf. on Robotics and Automation, Seoul, Korea, 2001.

[26] W. M. Shen and P. Will, "Docking in self-reconfigurable robots," presented at the *Int. Conf. on Intelligent Robots and Systems*, Maui, HI, 2001.

[27] K. Stoy, W. M. Shen, and P. Will, "Global locomotion from local interaction in self-reconfigurable robots," presented at the *7th Int. Conf. on Intelligent and Autonomous Systems*, Marina del Rey, CA, 2002.

[28] ——, "How to make a self-reconfigurable robot run?," presented at the *Joint Conf. of ICAMS and Autonomous Agents*, 2002.

[29] H. Bojinov, A. Casal, and T. Hogg, "Multiagent control of self-reconfigurable robots," presented at the *Int. Conf. on Multiagent Systems*, 2000.

[30] M. Yim, Y. Zhang, J. Lamping, and E. Mao, "Distributed control for 3D metamorphosis," *Autonom. Robots*, vol. 10, pp. 41–56, 2001.

[31] S. Vassilvitskii, M. Yim, and J. Suh, "A complete, local and parallel reconfiguration algorithm for cube style modular robots," presented at the *Int. Conf. on Robotics and Automation*, Washington, DC, 2002.

[32] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic decentralized control for a class of self-reconfigurable robots," presented at the *Int. Conf. on Robotics and Automation*, Washington, DC, 2002.

[33] H. Ihara and K. Mori, "Autonomous decentralized computer control systems," *IEEE Computer*, vol. 17, no. 8, pp. 57–66, 1984.

[34] K. Mori, S. Miyamoto, and H. Ihara, "Autonomous decentralized computer system and software structure," *Computer Syst. Sci. Eng.*, vol. 1, no. 1, pp. 17–22, 1985.

[35] K. Mori, "Autonomous decentralized system technologies and their application to train transport operation system," presented at the *High Integrity Software Conf.*, Albuquerque, NM, 1999.

[36] S. Fahlman, *Three Flavors of Parallelism*. Pittsburgh, PA: Carnegie Mellon Universtiy, 1982.

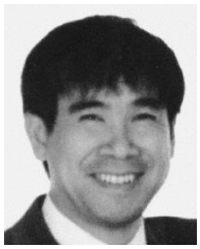
[37] R. C. Arkin, "Homeostatic control for a mobile robot: Dynamic replanning in hazardous environments," *J. Robot. Syst.*, vol. 9, no. 2, pp. 197–214, 1992.

[38] K. Sims, "Evolving virtual creatures," in *Proc. SIGGRAPH'94 Computer Graphics, Annu. Conf. Series*, 1994.

[39] *Knowledge Revolution Working Model 3D User's Manuals*, 1997. User-Manual.

[40] B. Salemi, W. M. Shen, and P. Will, "Distributed task selection in chain-type metamorphic robots," in *Proc. Int. Conf. on Robotics and Automation*, Taiwan, R.O.C., 2003. (submitted).

[41] H. A. Simon, *The Sciences of Artificial*. Cambridge, MA: MIT Press, 1996.



**Wei-Min Shen** (M'01) received the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, in 1989, under the Nobel Prize Winner, Prof. Herbert Simon.

He is currently the Associate Director of the University of Southern California's (USC) Center for Robotics and Embedded Systems (Los Angeles, CA), the Director of Information Sciences Institute (ISI) Polymorphic Robotics Laboratory, and a Research Assistant Professor in the ISI and Computer Science Department, USC. His research interests include artificial intelligence, robotics, and life science. He is the author of *Automonous Learning from the Environment* (San Francisco, CA: Freeman, 1994). He has chaired several international conferences and workshops in robotics, machine learning, and data mining, and has served on the editorial boards of two scientific books and one international journal. His research achievements have been reported by many news media, including CNN, PBS, Discovery channel, LA Times, BYTE, Chinese World Journal, and SCIENCES.

Dr. Shen has won several research awards in these fields such as the RoboCup World Championship Award in 1997, and the AAAI Robotics Competition Silver Medal Award in 1996.

**Behnam Salemi** received the B.S. degree in computer science from Shahid-Beheshti University, Tehran, Iran, in 1991, and the M.S. degree in computer science from the University of Southern California (USC), Los Angeles. He is currently working toward the Ph.D. degree in the Department of Computer Science, USC, and is a graduate research assistant at the Information Sciences Institute.

His research interests include self-reconfigurable robots, intelligent control, and autonomous distributed systems.

**Peter Will** (M'81) received the B.Sc. degree in electrical engineering and the Ph.D. degree in nonlinear control systems, in 1958 and 1960, respectively, both from the University of Aberdeen, Aberdeen, U.K.

He is a Fellow at the University of Southern California (USC)/Information Sciences Institute, Marina del Rey, CA. He is also a Research Professor in the Industrial and Systems Engineering and the Material Science Departments at USC, and is a Co-Founder of USC's Laboratory for Molecular Robotics. He spent 16 years with IBM's Yorktown Research Laboratory, where he led IBM's robotics group, seven years with Schlumberger, where he directed the AI laboratory, and five years with Hewlett-Packard Laboratories as the Director of the Measurement and Manufacturing Center. For six years, he was a Member of DARPA's ISAT study group. He has over 100 publications and 11 patents. He served as the Chair of three NSF Advisory Committees, the National Research Council's (NRC) Computer Science and Technology Board, and was Chair of the NRC Study on Information Technology in Manufacturing in 1994–1995.

In 1990, Dr. Will was awarded the Joseph E. Engelberger Prize in Robotics.