Distributed Behavior Collaboration for Self-Reconfigurable Robots

Behnam Salemi and Wei-Min Shen
Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292, USA
{salemi,shen}@isi.edu

Abstract— **This paper describes a distributed and decentralized approach for modules in a self-reconfigurable robot to select appropriate behaviors based on four factors: the current global task, the local topological location in the current configuration, the local state/sensor information, and the received messages from their neighbors. This approach does not assume any unique global identifiers for the modules, and is robust for reconfigurations of modules. The approach is enabled by the *extended neighbor topology* built upon a previous local topology representation and a hormone-inspired communication and control protocols. Experimental results on the CONRO robot have shown some unique features of this approach for the control of self-reconfigurable robots in general.**

Keywords- Self-reconfigurable robots,distributed control, behavior selection, distributed collaboration.

I. Introduction

A self-reconfigurable system is a special type of complex systems that can autonomously or manually rearrange its software and hardware components and adapt its configuration (such as shape, size, formation, structure, or organization) to accomplish difficult missions in dynamic, uncertain, and unanticipated environments. A self-reconfigurable system is typically made from a network of homogeneous or heterogeneous *reconfigurable modules* (or *agents*) that can autonomously change their physical or logical connections and rearrange their configurations. Self-reconfigurable robots [1-3] are examples of such systems that consist of many autonomous modules that have sensors, actuators, and computational resources. These modules are physically connected to each other in the form of a configuration network. Since the topology of the network may change from time to time, the controller of the robot must be distributed and decentralized to avoid single-point failures and communication bottleneck among modules. These modules must have some essential capabilities in order to perform complex tasks in dynamic and uncertain environments. These capabilities are: (1) distributed task negotiation [9, 10] – allowing modules to agree on a global task to perform, (2) distributed behavior collaboration – allowing modules to "translate" a global task into local behaviors of modules; (3) synchronization – allowing modules to perform local behaviors in a coordinated and timely fashion; and finally (4) topology monitor and discovery – allowing modules to detect changes/damages in the robot configuration and adopt behaviors or repair configurations. This paper is about the solutions to the second problem.

For distributed behavior collaboration, modules must know their topological location in the current configuration. For example, for an insect-like robot to walk, the modules in the legs must perform different actions than the modules in the body. All the leg modules must perform a set of coordinated behaviors to generate the insect walk gait. This task is more complicated in self-reconfigurable robots because modules cannot always assume to have unique global identifiers or addresses. For example, module M_i could be in arm in one configuration and in leg in another configuration. To present the topological location of modules, previous approaches were either centralized and assumed global identifiers for modules [4,12] or assumed simple neighboring relations such as binary occupancy code (yes or no for a neighbor) for chain-based robots [13] and binary occupancy grids in lattice-based robots [14]. These approaches only allow modules to know if they have neighbors or not and do not represent how neighbor modules are connected at the "connector" level. (e.g., a module's connector x is connected to the connector y of a neighboring module). Approaches that do consider the connector level, such as [5,6,7,8], only have considered information about the immediate neighbors. Such approaches are not powerful enough to support all possible behavior collaborations in the complex configurations.

This paper presents a new approach to distributed behavior collaboration based on the concept of "path" to represent extended neighborhood topology at the connector level. This allows modules to select appropriate local behaviors for a given global task in a given configuration. Specifically, a behavior of a module is a set of actions executed sequentially. In the insect-walk example, the relevant behaviors (among many existing local behaviors) of a module can be 'Leg-Lift-Forward Move', 'Leg-Down-Backward Move', 'Spine-Left Move', and 'Spine-Right Move'. The legged gait is the result of the coordinated performance of these behaviors. The collection of the behaviors of all modules over time is called a *group behavior* of the network. Although, any random combination of the modules' individual behaviors can be considered as a group behavior, most of them are not appropriate for accomplishing any global tasks. In the past, researchers have used machine-learning approaches to learn the mapping from the global task to individual behaviors [15], but they are limited to only fixed-shape and configuration specific robots.

This paper is organized as follows: Section 2 defines the problem of distributed behavior collaboration and uses the CONRO self-reconfigurable robot as an illustrative example;

Section 3 presents the basic idea of extended neighboring types; Section 4 describes the process of selecting local behaviors based on the extended neighboring types; Section 5 describes the D-BEST algorithm; Section 6 gives two examples on selecting behaviors for T-shape and snake-like configurations; Section 7 describes the experimental results on the CONRO robots; and Section 8 concludes the paper with future research directions.

II. DISTRIBUTED BEHAVIOR COLLABORATIOIN

The problem of distributed behavior collaboration can be defined as follows: Given a global task and a group behavior, selecting a correct set of local behaviors at each module and coordinate the selected behaviors to produce the desired global effects.

The problem is very challenging due to several reasons: relationships among modules are not static but change with configurations; the number of modules in the robot is not known; modules have no unique global identifiers or addresses; modules do not know the global configuration in advance, and can only communicate with immediate neighbors. Under these circumstances, a satisfactory solution to distributed behavior collaboration must be distributed. Modules must select behaviors through local communication, and the execution of the selected behaviors must be synchronized.

Formally, the problem of distributed behavior collaboration is a tuple (*P, Q, C, A, B, t, GB*), where *P* is a list of nodes, p_i; *Q* is the list of the internal state, q_i, associated with each node p_i, such that $i \in \{1,\ldots, N\}$; *C* is a list of labeled physical or logical links, c_j, such that $j \in \{locally\ unique\ labels\}$; *A* is a set of actions a_s a node can execute, such that $s \in \{1,\ldots, S\}$; *B* is a set of behaviors in the form of $b_m = (a_x, a_y\ a_z,\ldots)$, such that $m \in \{1,\ldots, M\}$; *t* is the global task given to all nodes, and *GB* is the desired group behavior in the form of behavior selection rules. These rules are mappings from nodes internal states to behaviors, $Q \rightarrow B$. The *configuration graph* of the network of modules is a graph consists of *P* nodes and *C* edges. A distributed behavior selection problem is solved if and only if $\beta = GB$, where $\beta = \beta \cup b_{pi}$, $i \in \{1,\ldots, N\}$; meaning that the union of the selected behaviors of all nodes over time is equal to the desired group behavior. Note that the size of the network is dynamic and unknown to the individual nodes; also the index numbers are only used for defining the problem and not used in the solution.

To illustrate the problem, we use the CONRO self-reconfigurable robot as an example. CONRO is a chain-type self-reconfigurable robot developed at USC/ISI (http://www.isi.edu/robots). Figure 1 shows the schematic views of CONRO module and a six-legged CONRO robot. Each CONRO module is autonomous and contains two batteries, one STAMP II-SX micro-controller, two servomotors, and four docking connectors for connecting with other modules. Each connector has a pair of infrared transmitters/receivers to support communication as well as docking guidance.

Each module has a set of open I/O ports so that various sensors for tilt, touch, acceleration, and miniature vision, can be installed dynamically. Each module has two Degrees Of Freedom: DOF1 for pitch (about 0-130° up and down) and

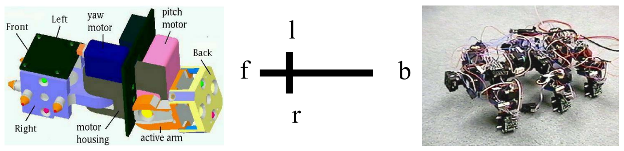


Figure 1: A CONRO module, the schematic view of one module, and a hexapod (insect) configuration with 9 modules.

DOF2 for yaw (about 0-130° left and right). The range of yaw and pitch of a module is divided to 255 steps. The internal state of each module includes the current values of the yaw, pitch of a module, and the number of the sent and received messages. The modules' *actions* consist of moving the two degrees of freedom to one of the 255 positions, attaching to or detaching from other modules, or sending messages to the communication links through the IR senders.

Modules can be connected together by their docking connectors. Docking connectors, located at either end of each module. At one end, labeled *back* (*b* for short), there is a female connector, consisting of two holes for accepting another module's docking pins. At the other end, three male connectors of two pins each are located on three sides of the module, labeled *left* (*l*), *right* (*r*) and *front* (*f*).

III. EXTENDED NEIGHBORHOOD TOPOLOGY

When modules in a self-reconfigurable robot have negotiated and decided on a global task, [10], they must then generate a group behavior to accomplish the task. A group behavior is the result of the coordinated performance of local behaviors of individual modules, while the local behaviors are selected based on the location of the modules relative to other modules. In a previous paper, [11], we represented the module's location in a configuration as the *type* of the module.

TABLE 1: LOCAL TOPOLOGICAL TYPES OF MODULES

	This Module					This Module				
	b	*f*	*r*	*l*	Type	*b*	*f*	*r*	*l*	Type
Connected to other modules					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

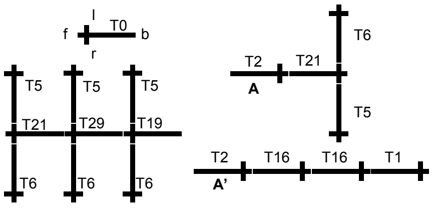


Figure 2: Immediate neighborhood topological types of CONRO modules in different configurations: a single module, a hexapod, a T-shape, and a snake.

Table 1 lists 32 types of CONRO module, which reflects how a module is connected to its immediate neighbors. Figure 2 shows some example types in various CONRO configurations.

The type information in Table 1 could provide modules with the necessary information to uniquely determine their location in most cases and select the appropriate local behaviors for the global task accordingly (see details in [11]). However, these types are not enough to guarantee determining modules' location in a complex configuration. For example, consider the T-shape and the snake configurations in Figure 2. The modules A, and A' are both of type T2, yet they must behave differently in the two different configuration. The module A' must perform a sinusoidal behavior in the snake configuration, while the A module must keep still in the T-shape "butter-fly" locomotion (i.e., the leg modules move in a cycle of up, left, down, and right, while the body modules keep still).

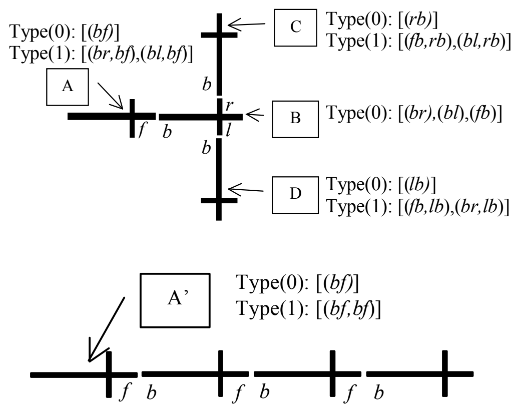


Figure 3: Examples of the extendable type(*n*).

To solve this problem, we extend module's type from the immediate neighborhood to neighbors that are *n* modules away. We call the extended type, type(*n*), and define it as how the active connection links of a given module are connected to the connection links of the modules of distance *n*. For example, in Figure 3, the type(0) for module A is [(*bf*)] because module B is the only one of distance zero from A, and the *b* connector of B is connected to the *f* connector of A. The type(1) of module A is [(*br,bf*),(*bl,bf*)] because this is how A is connected to module C and D, which are one module away (n = 1). Similarly, the type(0) of module A' is [(*bf*)] and its type(1) is [(*bf,bf*)]. Note that module B has only immediate neighbors (distance = 0) and therefore it has only type(0) information.

As we can see, although type(0) values of module A in the T-configuration and module A' in the snake configuration are the same, they have different type(1) value. It can be proven that using the extended types, modules can always uniquely identify themselves in a configuration as long as the labels of connection links of modules are locally unique. The proof is based on having unique path between any two nodes in a tree. As we will show next, modules can use the extended type information to select the appropriate behavior based on the given task.

It can be shown that the *type* definitions in [7,11] are special cases of the extended type and equivalent to type(0). In addition, global representation of the entire network for each agent is equivalent to [type(0),type(1), . . ., type(*d-1*)], where *d* is the diameter of the network.

The modules can discover their extended types dynamically and autonomously. The solution is based on the characteristics of hormone-inspired messages described in [3]. Each hormone message contains a path field that records a list of connector-pairs (ex. *bf*) through which the message has been propagated. When a module receives a hormone message with $|path|=m$, it will insert the path into its extended type(*m-1*) values. As more and more messages are received, the extended type information will be built up. Since messages are propagated through the network, each module will eventually build up the correct type values for itself.

IV. Selecting Local Behaviors Via Type(n) Values

The most straightforward approach for behavior collaboration in a modular system is the centralized 'gait control table' [yim94], in which a designated module, called the central controller, is given the information about behaviors of other modules in the form of a table. Each column of this table contains the sequence of actions that a module, identified by its Id, has to perform over time based on its location in the configuration (equivalent to the behavior of the module). The central controller job is to send each row of the table specifying the actions that all modules should perform at a time.

The 'gait control table' approach, however, is not an ideal approach for controlling the self-reconfigurable system for the following reasons: first, requiring the central controller to send actions to the rest of the modules in the configuration creates a communication bottleneck. In addition, if the central controller becomes faulty the entire system will be disabled. More importantly, when the network of modules restructures themselves, the pre-specified behaviors of the modules in the table might be valid anymore. The source of this difficulty is that modules do not know how their behaviors are chosen for them so that they can select new behaviors as they re-locate in the configuration.

Our approach for solving this problem is based on using the extended types to uniquely identify the location of modules in a configuration, and use them to select the correct local behaviors by the modules for the given global task. For example, as shown in Figure 4, for a quadruped to accomplish the 'Move forward' task, the 'front left leg' module and 'back right leg' will select 'Swing Backward' behavior, while the 'front right leg' module and the 'back left leg' module will select the 'Lift and Swing Forward' behavior and modules of types 'front spine' and 'back spine' will select 'bend left' and 'bend right', respectively. Figure 4b shows the robot after the modules have performed their selected behaviors. In general, the group behavior to accomplish "Move forward" consists of the following behaviors: the 'front left leg' and the 'back right leg' perform the 'Lift and Swing Backward' behavior, while the 'front right leg' and the 'back left leg' is performing the 'Swing Forward' behavior, and then the two groups switch their behaviors. In the next section we present a flexible algorithm called Distributed BEhavior SelecTion (D-BEST) for behavior selection, which is based on the extended neighboring types.

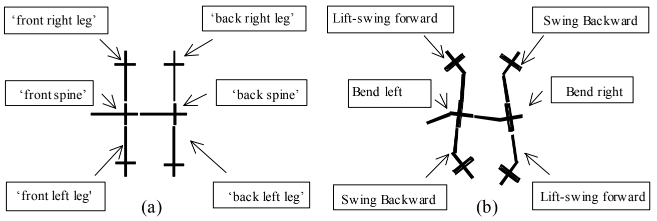
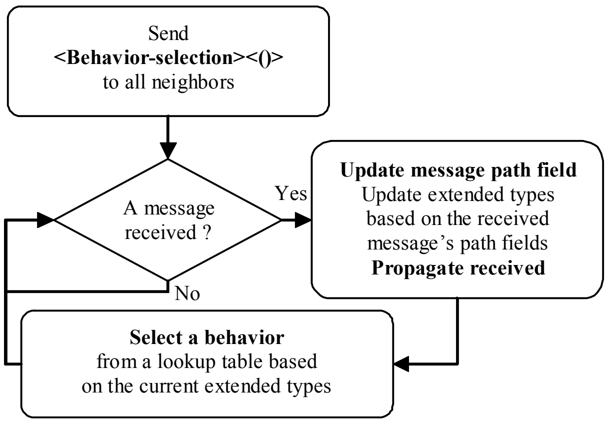


Figure 4: (a) The module types in a four-legged self-reconfigurable robot; (b) The selected local behaviors of each module for "move forward".

V. THE D-BEST ALGORITHM

Using the extended types as the condition for selecting behaviors will provide the modules with the information they require to autonomously collaborate to select new behaviors. Figure 5 illustrates the basic idea of the behavior collaboration based on the extended types. Initially, modules communicate their currently selected behaviors to their neighbors by sending hormone messages and wait for receiving new hormone message. This initial behavior could be a **Null** behavior. The communicated hormone message content consists of the type of the message, in this case of type **<Behavior-selection>**, and an initially empty path field, represented by **<(path)>**.

When a new hormone message is received, the module updates the path field of the message, updates its extended type based on the received path and propagates the message to its neighbors. Then it uses the current extended type to select a behavior from a lookup table representing the desired group behavior. This process will continue until all modules receive and propagate the initiated hormone messages.



Send
<Behavior-selection><()>
to all neighbors
A message received ?
Yes
No
Update message path field
Update extended types based on the received message's path fields
Propagate received
Select a behavior
from a lookup table based on the current extended types

Figure 5: Basic idea of behavior selection based on the extended types

Although this approach can dynamically adapt to the changes in the topology of the network, it has two problems. First, an initiated message from a module will be propagated to all other modules in the configuration. This means that the total number of communicated messages will be O(N^2), where N is the number of modules. The second problem is that when the diameter of the configuration is large, the size of the path field, and therefore the size of the message, will be large. This will considerably slow down the communication when the bandwidth is narrow.

To solve these two problems, we limit the maximum length of the path field in the messages. For example, if the maximum length of path is set to k, a message will stop being propagated after k hops. In this situation, if there are N agents in the network, and each agent has the average number of a active connectors, the number of communicated messages will be O(N) (since at most $a*N$ messages will be initiated and each message will be communicated k times therefore $k*a*N$ messages). The tradeoff of this solution is that the created extended type will be partial, and therefore might not be enough for some modules to select the correct local behaviors.

This problem can be solved by including the modules' selected behaviors in the communicated hormone messages and representing the group behavior as a set of *decision rules* based on the partial paths and received behaviors. In this situation, the content of the hormone messages and decision rules are shown in Figures 6a 6b, respectively.

<**Behavior-selection**><(**path**)><Selected behavior>
(**a**)

if (received *path* == X) **and**
(received *behavior* == Y)
then (select local behavior Z)
(**b**)

Figure 6: a) the format of the hormone messages. b) the format of the decision rules

The basic idea of this solution is that receiving the selected behavior of an extended neighbor gives an overview about the configuration of the module around that module, which combined with the received partial path can be used for selecting the correct behavior.

This control algorithm has some unique features that are different from previous approaches. Unlike the approaches based on the pre-assigned behaviors, this controller can adapt with the dynamic self-reconfiguration of the network, prevent communication bottleneck and is robust to individual modules failure. In addition, D-BEST is more flexible and powerful than the approaches for behavior selection based on immediate neighboring connection patterns as D-BEST uses both immediate and extended neighboring modules connection pattern for behavior selection.

It can be seen that the shorter the maximum size of the path results the smaller number of communicated message. This feature can be utilized at the design time of the decision rules for a desired group behavior in the following way. Starting from the smallest maximum length (k =0) the designer of the group behavior will write the rules that can uniquely select the correct behaviors for the modules. If there is ambiguity in selecting behaviors for the possible configurations, the k will be increased to provide the modules with more information such that the ambiguity is resolved. This characteristic of the D-BEST algorithm allows the number of the communicated message to be a function of the complexity of the desired group behavior and/or possible configurations.

VI. Example

In this section we present two examples of applying D-BEST algorithm for selecting behaviors in T-shape and snake-like configurations.

Figure 7 shows the rules for the Butterfly locomotion of a T-shape configuration. In this example Butterfly_Spine, CAT_0, Move_East and Move_West are different behaviors. In this example, the maximum path length is chosen to be zero, $k = 0$, meaning that immediate neighboring modules will not propagate the received messages. According to rules 1, if a module receives a message from one of it left or right connectors, it will be a spine module otherwise it can be a spine or a module in a snake configuration. Based on this rule, module B can select the correct behavior, Butterfly_Spine. However, module A does not know if it is a spine module or in the snake configuration. Rule 4 can resolves this issue by determining the module cannot be part of a snake configuration if the neighboring module B has selected Butterfly_Spine. Rules 2 and 3 will be used by the side legs to select the correct direction for their movements. If module A applies rule 1, it will consider the possibility of being part of a snake by selecting the CAT0 (sinusoidal motion starting from angle zero for caterpillar move). This selection will be corrected if at some point rule 4 is applicable.

1) **If** path = ((*bl*) **or** path = (*br*))
then select Butterfly_Spine
else select CAT_0

2) **If** path = (*rb*)
and behavior = Butterfly_Spine
then select Move_West

3) **If** path = (*lb*)
and behavior = Butterfly_Spine
then select Move_East

4) **If** (path = (*fb*) **or** path = (*bf*))
and behavior = Butterfly_Spine
then select Butterfly_Spine

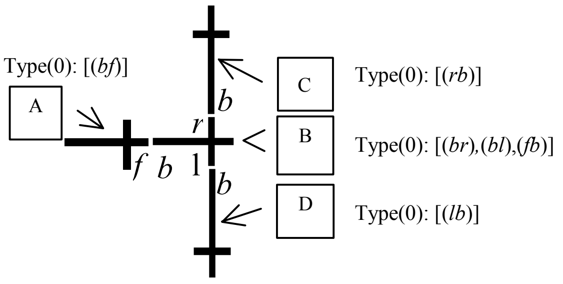


Figure 7: Decision rule for Butterfly locomotion.

In a situation that module A is actually part of snake configuration then the rules shown in the next example, Figure 8, will be applicable and the behaviors for the caterpillar move will be selected.

In the example of Figure 8, CAT0 to CAT120 are the behaviors for performing the sinusoidal motion starting form 0 to 120 degrees, respectively. In this example $k = 1$ meaning that the initiated messages will be sent to extended neighbors of distance one. We can see that in the case of $k = 1$, there will be an ambiguity in the extended types of modules C and D as they both have the same type(0) and type(1) while they are expected to perform different behaviors. However, using the received behaviors from the neighboring modules that have had selected correct behaviors without ambiguity, i.e. modules B and E, will allow these modules to select the correct behaviors.

It is important to notice that all the rules in figures 7 and 8 are part of a single database and in the behavior selection process all of them will be investigated. Hence, as the configuration of the network dynamically changes, the selected behaviors will be consistent with the current configuration.

If path = (*bf,bf*) **and** behavior = CAT_0
then select CAT_60

If path = (*bf*) **and** behavior = CAT_0
then select CAT_30

If path = (*bf,bf*) **and** behavior = CAT_30
then select CAT_90

If path = (*bf*) **and** behavior = CAT_30
then select CAT_60

If path = (*bf,bf*) **and** behavior = CAT_60
then select CAT_120

If path = (*bf*) **and** behavior = CAT_60
then select CAT_90

If path = (*bf,bf*) **and** behavior = CAT_90
then select CAT_150

If path = (*bf*) **and** behavior = CAT_90
then select CAT_120

If path = (*bf,bf*) **and** behavior = CAT_120
then select CAT_0

If path = (*bf*) **and** behavior = CAT_120
then select CAT_150

If behavior <> CAT * **then** select CAT_0

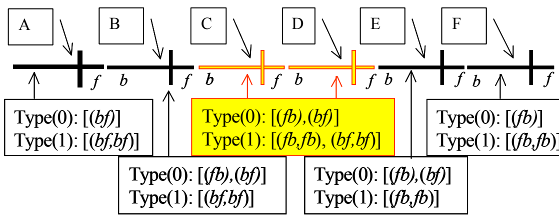


Figure 8: Decision rule for Caterpillar locomotion

VII. Experimental Results

We have implemented and tested the D-BEST algorithm in two sets of experiments. The first is on the real CONRO modules for locomotion. The second is on a simulated CONRO robot in a Newtonian mechanics simulation environment called Working Model 3D. Movies for these experiments can be found at http://www.isi.edu/robots.

All modules are loaded with the same control program. In some cases for different configurations, we have loaded different sets of rules. For economic reasons, the power of the modules is supplied independently through cables from an off-board power supplier. But all modules are running as autonomous systems without any off-line computational resources.

For the snake configuration, we have experimented with caterpillar movement with different lengths ranging from 1 module to 10 modules. With no modification of programs, all these configurations can move and snakes with more than 3 modules can move properly as caterpillar. The average speed of the caterpillar movements is approximately 30cm/minute. To test the ability of on-line reconfiguration, we have dynamically "cut" a 10-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 described approach is robust to changes in the length of the snake configuration.

For the legged configuration, we have experimented various configurations derived from a 6-leg robot. 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, these results can scale up to large configurations such as centipedes that have many legs.

In another experiment with the four-legged configuration, while the robot was executing the Walk task, we detached the two spine modules. The resulting configuration was two separate T-shape robots. In this situation each T-shape robot continues the locomotion by executing the Butterfly Stroke gait. Later, two T-shape robots were re-connected and the resulting four-legged robot re-initiated the four-legged Walking gait.

To test this approach for the self-reconfiguration task, we developed the rules for the 'Snake to T-shape' self-reconfiguration task. In this experiment, the self-reconfiguration task was given manually to one the middle module of a snake-shape robot consisting of seven modules. After completion of this task the behaviors for the T-shape butterfly gait was generated.

To test the autonomous reaction of the robot to the environmental stimulus, we installed two tilt-sensors on one of the modules in the snake configuration and loaded in all modules a set of rules for flipping back the robot to the normal orientation. Initially, a 'Caterpillar Move' task was manually given to the snake-shape robot. Then, while it was moving as a caterpillar, we manually pushed the snake to its side or flipped it upside down such that the tilt-sensors were activated. We observed that based on the output signals of the sensors, new tasks such as FlipLeft, FlipRight, or FlipOver was generated. Consequently, new behaviors were initiated to bend modules appropriately to flip the robot back to the correct orientation.

In parallel with the experiments on the real CONRO robot, we have also implemented the described approach on a simulated CONRO robot in a software Newtonian simulation environment called Working Model 3D. 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 controller 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.

VIII. Conclusion and Future Work

This paper described a distributed and decentralized solution for agents to select appropriate local behaviors based on their extended topological type and a given global task. The concept of extended type can guarantee modules to uniquely determine their location in the current configuration without assuming any global unique identifiers. The method is robust to configuration changes and scalable up to large systems. The proposed method has been tested on the physical CONRO self-reconfigurable robots and the results have demonstrated the desired properties described in the paper. As the future work, we will study the conditions for performing successful self-reconfiguration and locomotion tasks based on the received messages and develop a complete set of rules for performing all possible self-reconfiguration and locomotion tasks.

Acknowledgment

We are grateful that this research is sponsored by AFOSR under award numbers F49620-01-1-0020 and F49620-01-1-0441.

References

[1] Yim, M., Y. Zhang, D. Duff, *Modular Robots.* IEEE Spectrum, 2002

[2] Rus, D., Z. Butler, K. Kotay, M. Vona,, *Self-Reconfiguring Robots.* ACM Communication, 2002.

[3] Shen, W.-M., B. Salemi, and P. Will., *Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots.* IEEE Transaction on Robotics and Automation, 2002. 18(5): p. 700-712..

[4] Yim, M., *Locomotion with a unit-modular reconfigurable robot (Ph.D. Thesis)*, in *Department of Mechanical Engineering*. 1994, Stanford University.

[5] Stoy, K., WM. Shen and P. Will. Global Locomotion from Local Interaction in Self-reconfigurable robots. in IAS-7. 2002.

[6] Salemi, B., WM. Shen and P. Will. *Hormone Controlled Metamorphic Robots*. in *ICRA*. 2001.

[7] Stoy, K., Shen,WM., Will, P.,, Using Role-Based Control to Produce Locomotion in Chain-Type Self-Reconfigurable Robots. IEEE/ASME Transactions on Mechatronics, 2002. **7**(4): p. 410.

[8] Murata, S., E. Yoshida, H. Kurokawa, K. Tomita, S. Kokaji, *Self-Repairing Mechanical Systems.* Autonomous Robots, 2001. **10**: p. 7-2.

[9] Shen, W.-M., B. Salemi, and P. Will, Hormone-Inspired Adaptive Communication and Distributed Control for CONRO Self-Reconfigurable Robots, *IEEE Transactions on Robotics and Automation*, 18(5), October, 2002.

[10] Behnam Salemi, Peter Will, Wei-Min Shen, Distributed Task Negotiation in Self-Reconfigurable Robots, International Conference on Intelligent Robots and Systems. Las Vegas, October 2003.

[11] Salemi, B., WM. Shen and P. Will. *Hormone Controlled Metamorphic Robots*. in *ICRA*. 2001.

[12] H. Kurokawa, et. al., M-TRAN II: Metamorphism From a Four-Legged Walker to a Caterpillar. International Conference on Intelligent Robots and Systems. Las Vegas, October 2003.

[13] Y. Zhang et.al., Phase Automata: a programming model of locomotion gaits for scalable chai-type modular robots. International Conference on Intelligent Robots and Systems. Las Vegas, October 2003.

[14] R. Fitch, Z. Butler, D. Rus, Reconfiguration Planning for Heterogeneous Self-reconfiguring Robots. International Conference on Intelligent Robots and Systems. Las Vegas, October 2003.

[15] P. Maes and R. Brooks, "Learning to Coordinate Behaviors," Proceedings of AAAI-90: The American Conference on Artificial Intelligence, pp. 796-802 AAAI Press/MIT Press, Boston, MA, August 1990.