2009 IEEE International Conference on Robotics and Automation
Kobe International Conference Center
Kobe, Japan, May 12-17, 2009

Experiments with a ZigBee Wireless Communication System for Self-Reconfiguring Modular Robots

Robert Fitch and Ritesh Lal
ARC Center of Excellence for Autonomous Systems
Australian Centre for Field Robotics (ACFR)
University of Sydney, NSW Australia
{rfitch, rlal}@acfr.usyd.edu.au

***Abstract*— The problem of designing reliable low-cost communications systems to support decentralized algorithms is a major research challenge in self-reconfiguring modular robotics. In this paper we evaluate a communication system based on ZigBee, a wireless ad-hoc mesh networking standard. We present a 15-node system prototype and results from an experiment of 300 trials that measures system performance on a benchmark task. The benchmark we chose is the *connectivity problem* – how to maintain connectivity in the module graph during the disconnections and reconnections that occur during reconfiguration. We also provide full implementation details in pseudocode for our connectivity algorithm. Our results show that, despite its inherent scalability limitations, a ZigBee wireless system is feasible as a simple, low-cost communication system for self-reconfiguring modular robots.**

I. Introduction

Self-reconfiguring (SR) modular robots rely on robust communication. Decentralized planning and control algorithms are typically implemented using a message-passing model, and a fast, reliable communication infrastructure is critical to their proper performance. To conform to the overall design goals of SR robots [1], communications systems must also be low-cost.

Hardware implementations of SR robots typically build communication components into module faces or the connector mechanism itself using infrared (IR) technology or a physical (essentially wired) link. IR-based systems are notoriously difficult to implement reliably, mainly due to problems with crosstalk [2], [3]. Wired links are fast but complicate the connector design problem.

A communication system based on a wireless mesh network is appealing because it is simple and reliable, and can be easily integrated into existing or new module hardware designs because it does not require components to be embedded within module faces or connection mechanisms. Commercial implementations of wireless mesh systems are inexpensive and readily available. Bluetooth and Zigbee [4] are the two main protocol standards. We chose to investigate Zigbee because, unlike BlueTooth, it can operate without a central controller, is self-healing, and does not place tight constraints on network size. Bluetooth is limited to small networks and requires the presence of a central controlling node.

The fundamental disadvantage of a wireless mesh system is that modules cannot communicate in parallel. This limitation implies that decentralized algorithms will not scale to large numbers (thousands or millions) of modules. However, this might not be a problem in certain applications. For example, we are interested in building a fieldable modular robot with significant sensing capabilities [5]. In this application, separate groups of modules work together to perform an imaging task. Each group could be small enough to work within scalability constraints, and utilize multiple channels to avoid contention. Also, a multi-radio, multi-channel scheme could potentially be used to increase the maximum group size [6].

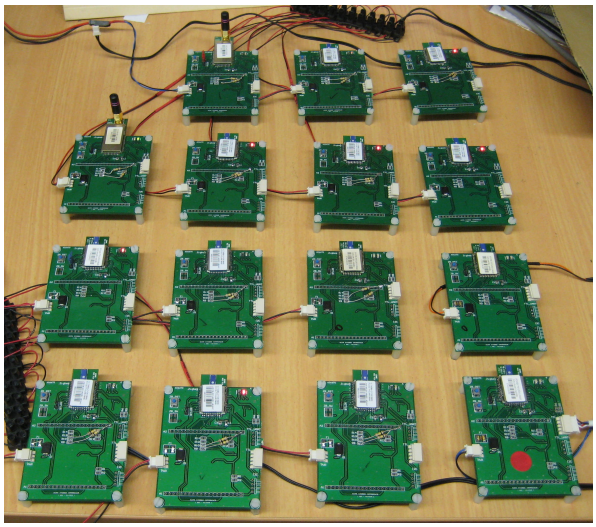

Fig. 1: Our 15-node ZigBee-based communication system.

In this paper, we are interested in quantifying the real-time performance of a ZigBee-based system for small-scale SR robots (tens of modules) under a typical communication load. We present a 15-node prototype, shown in Fig. 1, and results from experiments that evaluate the performance of this system on a benchmark decentralized planning task. We chose our recent parallel connectivity check algorithm [7] as a benchmark since it solves a problem common to all SR robots and is strenuous – every module communicates within its local neighborhood in parallel and as fast as possible. Our implementation assumes that modules have globally unique IDs and all modules know the identities of their neighbors.

We fully describe the decentralized implementation of the algorithm, including pseudocode. Our results demonstrate that, although scalability issues are readily observable, the system is feasible for small groups of modules and is low-cost, low-complexity, and is a convenient platform for implementing decentralized algorithms.

Our paper is organized as follows. We discuss related work and a comparison with Bluetooth in Sec. II. In Sec. III-B, we present our hardware prototype, and in Sec. IV we present implementation details of the connectivity algorithm. We then describe our experiments and results in Sec. V and conclude in Sec. VI.

II. Related Work

Our work builds on a rich history of research in communications systems and decentralized algorithms for modular robots. See the recent paper by Yim et al. [1] for a survey of decentralized planning and control schemes in the literature.

Two major issues in communications research are: 1) the tradeoff between local and global communication, and 2) choice of technologies to implement the physical layer. The local vs. global question has been well-studied; for a nice discussion see Garcia, Stoy, and Lyder [8], who propose a novel hybrid system.

The majority of SR robots use either an IR link [9], [10], [3], [11] or a wired link using a physical connection [12] as the primary communication medium for planning and control. Unless much care is taken in designing proper shielding, IR-based systems are prone to the major problem of crosstalk [2], [9]. Physical connections must be built into the inter-module connection system and add complexity to this already-difficult design problem. Avoiding both issues is a major motivation for us in investigating wireless RF systems such as ZigBee.

A small number of more recent systems have utilized Bluetooth, the closest alternative to ZigBee. Most use Bluetooth in an auxiliary capacity [11], [12], [13], although the YaMor robot has implemented a significant Bluetooth scheme with great success by designing a custom protocol to increase the allowable network size [14]. The ZigBee standard is designed to implement large networks, albeit at the cost of a lower data rate. We discuss this issue further in Sec. III-B.

The recently presented Em-Cube robot [15] utilizes ZigBee, but to the best of our knowledge this paper is the first extensive performance evaluation of a ZigBee system for SR robots applications.

III. Hardware Design

We constructed a system of 15 ZigBee modules for experimental purposes. This section discusses advantages and disadvantages of the ZigBee standard, and presents our hardware design.

A. ZigBee

ZigBee is a standard developed by an industrial alliance for wireless radio networks in monitoring and control applications [4]. This standard builds on the IEEE 802.15.4 physical/datalink layer providing flexible network topologies, intelligent message routing and better communications security. ZigBee operates in the unlicensed radio frequency (RF) bands – 868, 915 and 2400 MHz. These three bands provide a total of 27 channels with RF data rates of 20, 40 and 250 kbps, respectively [16]. The 2.4 GHz band is widely used because it is unlicensed worldwide, has the highest data rate with more channel options, and consumes the lowest power. A standard ZigBee device (with around 0 dBm output power) is purported to achieve transmission range of 30 m indoors and over 200 m in open space.

Star, tree and mesh network topologies can be realized with ZigBee. A network comprises a coordinator, routers

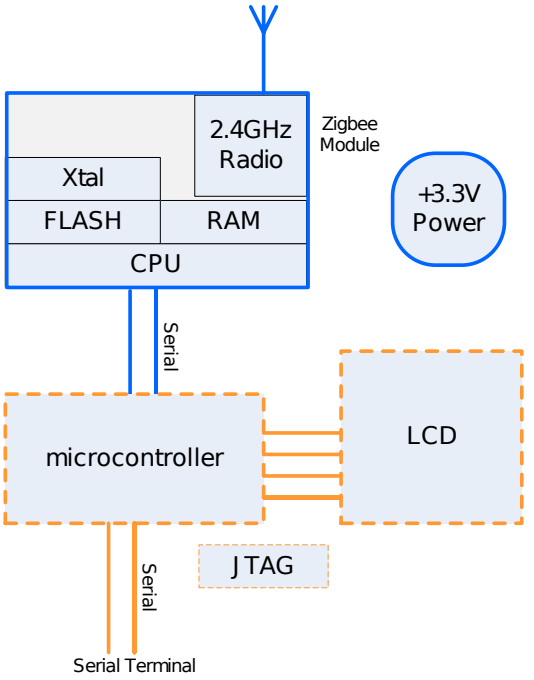

Fig. 2: Block diagram of our system. Components shown with dashed lines were not used for experiments in this paper.

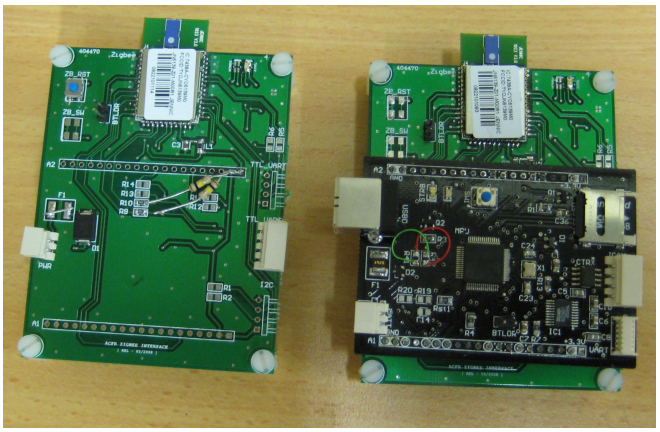

Fig. 3: Single module, with and without optional processor board.

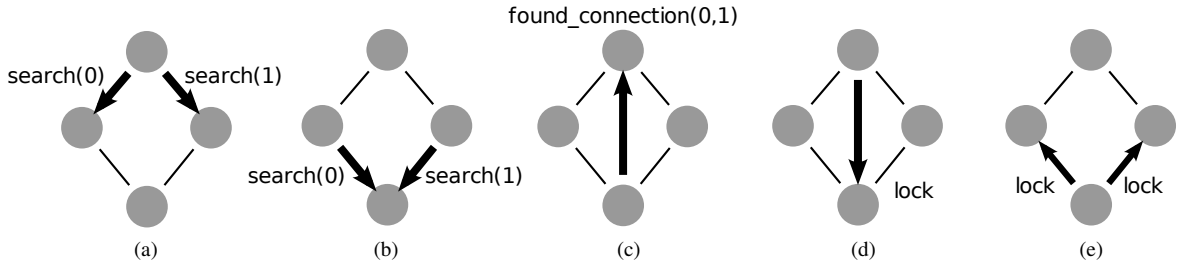

Fig. 4: Sample execution of connectivity algorithm on a 4-node network. Dark arrows indicate message-passing. Text labels indicate message type.

and end devices. The coordinator plays a role only when the network starts and is not required after all nodes have joined. Messages in a ZigBee network can be point-to-point, broadcast or multicast. However, point-to-point messages cannot be sent in parallel since the network uses a single RF channel. ZigBee is designed to be tolerant to RF interference and employs a few mechanisms to provide robust communications. These techniques include Carrier Sense, Multiple Access with Collision Avoidance (CSMA-CA), message acknowledgements, and alternative routes [16].

The closest competitor to ZigBee in terms of wireless communication technology is Bluetooth. Although Bluetooth claims a much faster data rate (1 Mbps vs. 250 kbps), ZigBee specifies a longer transmission range and is specifically designed for low power consumption [17]. The big disadvantage of Bluetooth in modular robotics applications is that it requires a central coordinator and is limited to small networks. ZigBee does not have this limitation. As discussed in Sec. II, one research group has developed a custom Scatternet protocol that permits Bluetooth use in larger networks, but the simplicity of an off-the-shelf, low-cost (around $20 USD in 2008) implementation lead us to investigate ZigBee.

B. Hardware Implementation

Hardware implementation of the ZigBee standard is available in a number of forms. We chose the JN5139 model produced by Jennic. This module has a 32-bit CPU, 96 kB of RAM, 192 kB of flash memory, and an antenna mounted on a small printed circuit board. This on-board processing allows simple applications to be run on the ZigBee module itself, without an external processor. Because we are interested in building a system with significant signal-processing capability, we included the facility to add a separate processor (not used in the experiments in this paper). The block diagram in Fig. 2 shows our current system design, and also features to be incorporated in the next version. These features include an LCD display to be used in debugging application code. Fig. 3 shows an example with and without the optional external processing.

Algorithm 1 Connectivity algorithm summary

```
Search for connecting cycles (search is depth-limited)
if all pairs of neighbors connected then
    lock connecting cycles
    return true
else
    repeat search with increased depth limit
end if

if received search message with higher priority then
    terminate search
    return false
end if
```

TABLE I: Message types used in connectivity algorithm.

Message	Data	Description
START	-	sent to begin algorithm
SEARCH	$rootID, color,$ $parent, depth$	outgoing search message
FOUND_CONN	$color1, color2$	sent to announce connecting cycle
SEARCH_RETURN	$rootID, color$	incoming search message
CLEAR_LABEL	$rootID$	clears color label
LOCK	-	locks a module

IV. Implementation of Connectivity Algorithm

Before any reconfiguration step can be performed in an SR robot, the system must ensure that module disconnection does not result in a disconnection of the module connectivity graph. This is known as the *connectivity problem*. When moving modules serially, this problem can easily be solved with basic graph algorithms. However, finding a *set* of mobile modules is more difficult. We previously presented a parallel decentralized algorithm that finds a maximal set of mobile modules [7]. Here we present detailed pseudocode for implementing this algorithm in a hardware system such as our ZigBee network.

The algorithm is summarized in Algorithm 1 for convenience. As search proceeds through module network, modules are labeled as they are visited by the search. The initiating module chooses a unique label for each neighbor. When the search visits a module already labeled (with a label different from its own), this signifies that a path has been found connecting a pair of neighbors. This path is called a *connecting cycle*. The module is mobile when all neighbors are connected by a connecting cycle and modules along this path are successfully *locked*. While locked, a module must

Algorithm 2 Decentralized implementation of connectivity algorithm

State:
ID, unique identifier of this module
neighbors, list of neighbor IDs
colors, list of current labels
parents, list of parent-pointers associated with colors
ds, disjoint set data structure
locks, number of modules currently locking me
depthLimit, current limit of search depth (in hops)

Message Handlers:
procedure HANDLE_START
send SEARCH to all neighbors, with unique color labels
end procedure

procedure HANDLE_SEARCH($rootID, label, parent, d$)
if executing search and $rootID$ has lower priority **or** depth limit exceeded **then**
send SEARCH_RETURN to parent and return
else
store $label$ in $colors$ and $fromID$ in $parents$
increment depth d and forward SEARCH message to neighbors
end if
end procedure

procedure HANDLE_FOUNDCONN($color1, color2$)
store color pair $color1, color2$ in ds
if ds has all neighbors in one set, send LOCK message
end procedure

procedure HANDLE_SEARCHRETURN($rootID, color$)
forward message to neighbor stored in $parents$
end procedure

procedure HANDLE_CLEARLABEL($rootID$)
remove color label from $colors$ for $rootID$
end procedure

procedure HANDLE_LOCK
increment $locks$ variable
end procedure

not perform a disconnection operation. All modules execute this algorithm in parallel; in order to avoid deadlock, if a module is visited by a search message from a module with higher priority, the module terminates its own search and allows itself to be locked if required. A module terminates its search by clearing the labels of all modules visited, and may restart its own search when all labels and locks are cleared.

It can be difficult to visualize the execution pattern of asynchronous algorithms, so we illustrate a simple example in Fig. 4. Messages are drawn in a synchronized manner, but in hardware these messages are transmitted asynchronously. This example shows a small network, but execution in a larger network can be imagined by extending the "reach" of messages at each step.

To implement this algorithm in hardware, we define a set of message types and associated code. Code executed upon receipt of a message is known as a *message handler*. The set of message types used in our implementation is listed in Table I and message handlers are listed in Algorithm 2. We also list the state variables required (stored in local memory). This pseudocode roughly follows the style of Lynch [18]. The data structure used to manage pairs of connected neighbors is a *disjoint set* [19].

Memory requirements for this algorithm scale with the number of simultaneous searches in which a given module can expect to participate. For example, in a small network such as that shown in Fig. 4 each module will receive a search message from all nodes in the network and the memory requirements would be $O(n)$, where n is the number of nodes. However, in a million-module network each module will most likely only receive search messages from a local neighborhood of some fixed size. Specific implementations can thus optimize memory use accordingly. In our 15-node network, we allocate $colors$ and $parents$ data structures sufficient to hold data for the full network.

V. Experiments and Results

To evaluate the feasibility of our system in its intended application, we performed a number of experiments. First, we conducted basic tests to quantify message delays and the maximum available throughput. We then executed Algorithm 2 multiple times while varying search depth and robot size to assess the real-time performance of the system on a benchmark task. All experiments were performed in a WiFi-dense environment (the ACFR field lab). Our application code executed directly on the ZigBee modules with no external processing or memory resources. Results are presented in this section.

A. Basic Tests

To measure message latency for small messages, we performed a "ping" test. In this test, an 8-byte message was sent along a network of 4 nodes connected in a ring topology. We chose 8-bytes because this is the message size

TABLE II: Measured time-of-flight per message and corresponding data rate. Each trial consists of sending 13,312 80-byte messages between two nodes.

Trial	Mean (ms)	Min (ms)	Max (ms)	Data rate (kbps)
1	11	8	70	54.3
2	11	8	54	54.2
3	11	8	76	54.1
4	11	8	62	54.0
5	11	8	45	54.3

TABLE III: Observed message counts for execution of connectivity algorithm in simulation and hardware with varying network size and search depth. Averages are taken over 20 trials.

	Message count, depth 2		Message count, depth 4		Message count, depth 8	
Modules	Simulation	Hardware	Simulation	Hardware	Simulation	Hardware
4	43.2	56.4	46.4	57.2	-	-
5	48.9	87.9	54.3	107.0	55.4	99.6
6	95.9	104.7	119.3	123.9	116.0	117.6
9	165.1	234.7	279.8	348.5	309.9	322.1
12	235.1	275.4	379.8	529.9	500.3	531.8
15	307.3	539.2	-	-	-	-

used later in our benchmark implementation. We measured time-of-flight between a message leaving and returning to an originating node. For 1000 trials, we observed average, minimum, and maximum time-of-flight values of 28 ms, 23 ms, and 50 ms respectively. With 4 hops this gives us an average of $28/4 = 7$ ms per hop. These observations confirm other measurements reported in the literature [20].

Next we performed a bandwidth test to determine the maximum data rate the system can achieve. We measured bandwidth by transmitting an arbitrarily large amount (1 MB) of random data between two nodes in messages of maximum size allowed by ZigBee (80-bytes), waiting for an acknowledgement from the destination node after each transmission. Table II lists results for five trials of this test.

Finally, we measured power consumption for one node during the tests above. We observed power use of 0.6 W (76 mA) at 8 V while the radio was active, and 0.56 W (70 mA) at 8 V while idle.

B. Connectivity Algorithm

To assess the real-time performance of the system, we implemented Algorithm 2 as a benchmark task.We performed a total of 300 trials with different combinations of network size (ranging from 4 to 15 modules) and algorithm parameters (search depths of 2, 4, and 8) with 20 trials per combination. Our experimental setup was as follows. The hardware was arranged on a flat surface as shown earlier in Fig. 1. Radio antennas were separated by approximately 10 cm. No effort was made to suppress interference from nearby WiFi sources. One module was fitted with a button and was connected to a desktop computer via a serial link. This allowed the special module to display data on the desktop monitor. A separate ZigBee "sniffer" was connected to the desktop computer to record network traffic (but was not part of the network).

The algorithm was implemented in the C language and ran directly on the ZigBee module, sharing processing and memory resources with the ZigBee stack. Each message was handled on arrival, but outgoing messages were queued in our application code and only sent after acknowledgement of the previous sent message. We observed that the ZigBee stack resends unacknowledged messages after 16 ms, but sometimes terminates retransmission without explicit notification. We therefore implemented a second layer of retransmission from our outgoing queue at 1000-ms intervals.

Each trial was manually initiated by pressing the button on the special module, which broadcast a START message to all nodes. Mobile modules signalled success by communicating execution time to the special module. Network traffic was measured using the sniffer system. The algorithm was configured to use a fixed search depth throughout execution. Search depth and network size were varied across trials. We also implemented the algorithm in simulation, using nearly identical C code, for comparison. Each simulated node was permitted to process a random number of messages, and nodes were chosen for execution at random.

Resulting message counts are listed in Table III. We see increased messages for increasing values of both network size and search depth as expected. The discrepancy in message counts between hardware and simulation arose due to message retries – the sniffer counted all message traffic in the network. Because we observed frequent out-of-memory exceptions from the ZigBee stack, we attribute this retry rate to memory limitations in our hardware. Our application code used memory otherwise available to the stack.

The high message retry rate is also visible in the real-time data, plotted in Fig. 5. This figure plots average execution time versus the number of modules in the system. Time increases with network size due to increased message counts, and also because of the algorithm's prioritization scheme. The highest priority module will temporarily prevent neighbors from executing their own search until it completes. Lower priority modules then restart their search. Increasing search depth worsens this effect, although at the highest depth the time decreases because the long connecting cycles permanently lock more modules. Even though search is in breadth-first order, asynchronous message-passing means that short paths are not guaranteed.

VI. Conclusions and Future Work

The main conclusion we draw from our results is that a ZigBee system is simple, low-cost, low-power, and robust, at the expense of data rate and scalability. We have shown experimentally that an off-the-shelf ZigBee system is a good alternative to IR for small robots with tens of modules. This

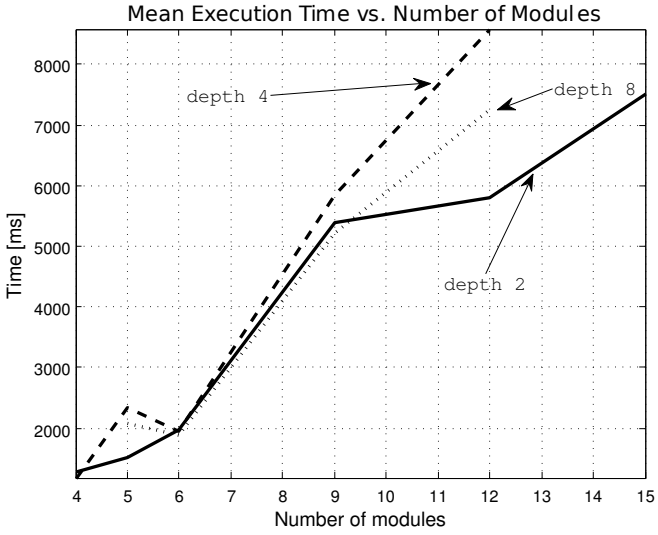

Fig. 5: Plot of mean execution time (clock time) of mobile modules versus network size. Time was measured between the start of a trial and algorithm success. Execution time for non-mobile (locked) modules was not measured. The algorithm was executed with fixed search depths of 2, 4, and 8 hops with network sizes of 4, 5, 6, 9, 12, and 15 modules. Mean values are taken over 20 trials.

type of system can be useful in certain applications, such as multiple disconnected groups of modules working in parallel.

The high message latency shown in our results is largely due to memory limitations in our choice of hardware. The ZigBee stack had to compete with our application code for memory, resulting in message loss with increased network traffic. The application then was forced to resend messages, which was the major source of delay. Use of an external processor will remove this memory contention and we expect real-time performance to improve.

Scalability is a more significant problem. We plan to investigate the idea of a multi-radio, multi-channel system to improve maximum network size while retaining the other advantages we have discussed. This approach will still only scale linearly with increased available bandwidth, but nevertheless should comfortably support our target system scale.

Other relevant issues not addressed in this paper that require experimental validation include transmission range (can modules communicate over a distance) and potential interference from mechanical elements in module hardware. We plan to investigate these questions in future work by performing experiments with real SR robots.

The problem of dynamic neighbor discovery was not addressed in this work. This is an issue we plan to investigate in future work by studying module self-localization. A module can discover the identity of a neighbor by sending a query that includes the neighbor's expected location.

One lesson learned from our experiments is that algorithms must be immune to multiple message delivery. Even without application-level retries, the ZigBee protocol will sometimes successfully deliver a message multiple times. An unexpected lesson we learned was the value of a packet sniffer in developing and debugging code. This tool was invaluable in implementing the asynchronous connectivity algorithm quickly and successfully.

ACKNOWLEDGMENT

This work is supported by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales (NSW) State Government.

REFERENCES

[1] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular self-reconfigurable robot systems: Challenges and opportunties for the future," *IEEE Robot. Automat. Mag.*, vol. 14, no. 1, pp. 43–52, March 2007.

[2] D. Christensen, D., U. Schultz, and K. Stoy, "Neighbor detection and crosstalk elimination in self-reconfigurable robots," in *RoboComm '07: Proceedings of the 1st international conference on Robot communication and coordination*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 1–8.

[3] K. Gilpin, K. Kotay, D. Rus, and I. Vasilescu, "Miche: Modular shape formation by self-disassembly," *Int. J. Rob. Res.*, vol. 27, no. 3-4, pp. 345–372, 2008.

[4] Z. Alliance, "http://www.zigbee.org/en/," WebSite, September 2008.

[5] J. Gomez and G. Brooker, "Opportunities for imaging in distributed robotics applications with ultra-wideband radars," in *3rd International Conference on Sensing Technology (ICST2008)*, 2008, pp. 15–20.

[6] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *Proc. of MobiCom '04*. New York, NY, USA: ACM, 2004, pp. 114–128.

[7] R. Fitch and Z. Butler, "Million module march: Scalable locomotion for large self-reconfiguring robots," *Int. J. Rob. Res.*, vol. 27, no. 3-4, pp. 331–343, 2008.

[8] R. Garcia, K. Stoy, D., and A. Lyder, "A self-reconfigurable communication network for modular robots," in *RoboComm '07: Proceedings of the 1st international conference on Robot communication and coordination*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 1–8.

[9] Z. Butler, R. Fitch, and D. Rus, "Distributed control for unit-compressible robots: Goal-recognition, locomotion and splitting," *IEEE/ASME Trans. Mechatron.*, vol. 7, no. 4, pp. 418–30, Dec. 2002.

[10] B. Salemi, M. Moll, and W.-M. Shen, "SUPERBOT: A deployable, multi-functional, and modular self-reconfigurable robotic system," in *Proc. of IEEE IROS*, Beijing, China, Oct. 2006.

[11] M. Yim, B. Shirmohammadi, J. Sastra, M. Park, M. Dugan, and C. Taylor, "Towards robotic self-reassembly after explosion," in *Proc. of IEEE IROS*, 2007, pp. 2767–2772.

[12] H. Kurokawa, K. Tomita, A. Kamimura, S. Kokaji, T. Hasuo, and S. Murata, "Self-reconfigurable modular robot M-TRAN: distributed control and communication," in *RoboComm '07: Proceedings of the 1st international conference on Robot communication and coordination*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 1–7.

[13] C. Detweiler, M. Vona, Y. Yoon, S.-K. Yun, and D. Rus, "Self-assembling mobile linkages," *IEEE Robot. Automat. Mag.*, vol. 14, no. 4, pp. 45–55, Dec. 2007.

[14] A. Sproewitz, R. Moeckel, J. Maye, and A. J. Ijspeert, "Learning to move in modular robots using central pattern generators and online optimization," *Int. J. Rob. Res.*, vol. 27, no. 3-4, pp. 423–443, 2008.

[15] B. An, "Em-cube: cube-shaped, self-reconfigurable robots sliding on structure surfaces," in *Proc. of IEEE ICRA*, 2008, pp. 3149–3155.

[16] Jennic, "http://www.jennic.com/," WebSite, September 2008.

[17] S. T. Group, "http://www.stg.com/wireless/index.html," WebSite, September 2008.

[18] N. Lynch, *Distributed Algorithms*. San Francisco: Morgan Kaufmann Publishers, 1996.

[19] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: The MIT Press, 2001.

[20] T. R. Burchfield, S. Venkatesan, and D. Weiner, "Maximizing throughput in zigbee wireless networks through analysis, simulations and implementations," in *Proc. of the International Workshop on Localized Algorithms and Protocols for Wireless Sensor Networks*, 2007, pp. 15–29.