Distributed Control and Communication Fault Tolerance for the CKBot

Michael Park, Mark Yim

GRASP Laboratory
University of Pennsylvania
3330 Walnut Street
Philadelphia, PA 19104
USA
parkmich@grasp.upenn.edu
yim@grasp.upenn.edu

Abstract—*In this paper we present a method for distributed fault tolerance in a modular robotic system. We describe an implementation of this method on the CKBot system. In particular, we broadcast infrared (IR) signals to modules which collaboratively vote on a majority course of action. Various gait selections for a 7 module caterpillar and a 16 module quadruped with a faulty subset of IR receivers have been verified to demonstrate the algorithm's robustness. We conclude the paper with a discussion of modes of fault tolerances and this method's applicability to other modular robotic systems.*

Index Terms—*Modular robots, fault tolerance*

1. Introduction

Modular robotic systems are comprised of modules that can be arranged in different ways to form various shapes. Modular *self-reconfigurable* robotic systems can rearrange their own modules. These different arrangements allow these systems to be very versatile and adaptive to suit a wide variety of situations.

Since these systems are designed to be reconfigured, each module is in some sense a self-contained unit. Most of the modules in these systems come equipped with their own embedded processors [1], [2], [3], [4], [5], [6], [7], [8]. Individual processors on each module allow systems to be more readily scaled upward since computational resources increases with number of modules. Processes run locally near locations where actions need to take place. For instance, computation for motor control and sensor processing can occur on the modules which share the motor and sensors. However, complications start to occur when groups of modules must coordinate their actions (e.g., a chain of modules which form a robot arm must coordinate their motions to control the end effector path).

Control may be centralized or decentralized [9], [10], [11], [12], [13], [14]. Often times, coordinated tasks such as end effector kinematic control require some synchronization and communication between modules. Centralized control, where an off-board PC or designated on-board processor orchestrates the overall execution of tasks has the advantage of being simple to implement and often is sufficient for accomplishing a variety of tasks. For example, on CKBot, the system used in the following work, we have demonstrated dynamic rolling [15] and [16] bio-inspired gaits which use centralized controllers.

However, central controllers have a key weakness in that they are inherently fault intolerant. If the central controller breaks, then the system as a whole is lost without its leader. This shortcoming effectively limits a purely centralized system's ability to realize the goal of robustness for modular robots. In an effort to study and develop a more robust control scheme, we devised a decentralized approach for controlling clusters of CKBot modules.

This paper focuses on fault tolerant distributed control through collective decision making. By sharing information and making decisions as a group, the system is more robust in the case of failures (in this work, IR communication reception). Our basic approach is for a group of modules all observing one signal to share the observed data and determine the best decision, even if some of the modules have faulty IR receivers or communication failures.

There is a variety of work related to robust control for modular self-reconfigurable robots using distributed control. In [17], [10] global behaviors such as locomotion are created through local cyclic behaviors. No central controller exists and synchronization occurs through local message passing. This system also handles changes in the number of modules (additions or deletions) during the course of its runtime. This style of robustness exploits the cyclic nature of gaits and is difficult to expand to more general, high-level control tasks.

Other related work in the area of modular robot robustness is self-repair, such as [18], [19], [20], [21] where robot modules repair themselves, by putting themselves back together or disposing of failing modules. In this case, self-repair occurs after failure, whereas the proposed work intends to deal with failures in module communication resulting in more robust information processing.

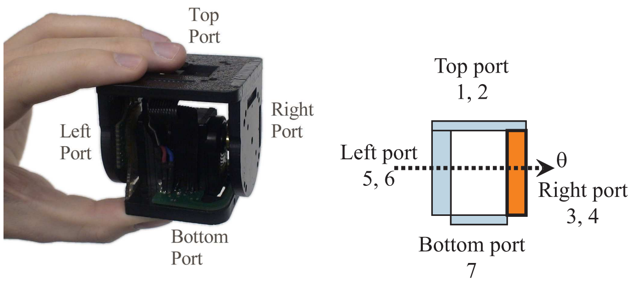

Fig. 1. A CKBot module with a schematic representation. The arrow indicates the rotational axis and the numbers designate the locations of the seven IR ports.

2. Experimental System

2.1. CKBot Module

For this work, we use the Connector Kinetic roBot (CKBot). Fig. 1 shows a CKBot module with four connection ports and the corresponding two-dimensional representation. Each port except the bottom port has two IR transmitter (TX) and receiver (RX) pairs. Fig. 2 shows the layout of the seven IR pairs. These IR pairs can be used for a variety of applications; in this work, the RX LEDS are used to receive messages from high-intensity IR broadcasting sources. In other works, the IR LEDS have been used for local (neighbor-to-neighbor) communication, ground contact detection [15], configuration recognition [22], and docking detection [20].

Each module has a 180° rotational degree-of-freedom, powered by a high-end RC servo. On-board processing is done with a Microchip PIC2680 with on-chip Control Area Network (CAN) capability [23]. For connectivity, two modules are connected physically and electrically (modules share power (battery or DC power source) and CAN communication) by inserting a 20 pin header between two modules. In this work, they are arranged in various configurations then firmly attached with screws, though in other scenarios, magnet faces are employed [20] to allow automatic reconfiguration. All modules have their own unique ID and can communicate to one another on the CAN. Therefore, any module can talk to any other in a given connected system.

Each CKBot module's processor independently handles its servo position actuation, CAN communication and IR data detection.

2.2. IR Broadcaster

In this work, a configuration of CKBot modules receives commands from arrays of high-intensity IR LEDs that blink commands in unison. In this way, modules can individually receive IR messages and share their observations with one another via the CAN. The particular data broadcasted from the IR transmitter boards is designated by a user at a PC which inputs commands through a simple graphical user interface [24].

A reader familiar with IR signaling may have wondered why we chose to use so many emitters in unison (instead of a few IR emitters similar to those used for television remote controls). The answer is simply because the IR receivers on the modules were designed primarily for module-to-module local communication and very close proximity retro-reflective distance measurements. As such, large arrays of LED emitters were used to produce the IR intensity required for the modules to receive the trasmitted data over the given range and distances.

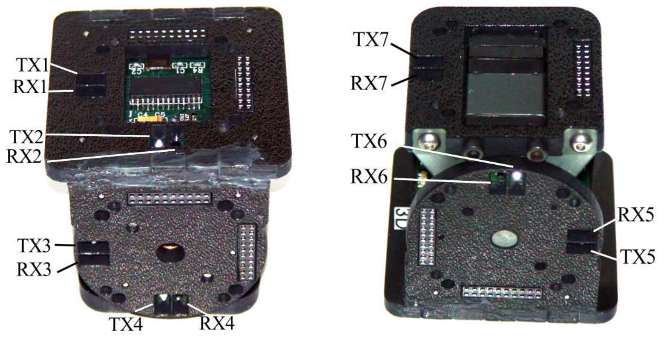

Fig. 2. Layout of the seven IR transmitter and receiver pairs on the four faces of a CKBot module.

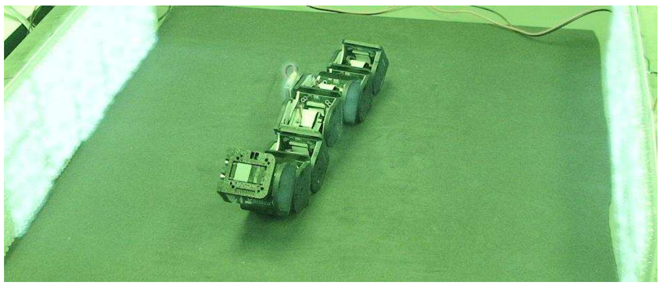

Fig. 3. Seven module caterpillar receiving IR signals from broadcasting boards. Digital camera in night vision mode captures the illuminated IR LEDS.

3. Distributed IR Fault Tolerance

To demonstrate distributed fault tolerance, the CKBot system is designed so that modules in a given configuration listen to globally broadcasted IR signals and subsequently communicate with one another to decide on actions, even in the presence of faulty modules. Experiments consist of configurations flooded with IR signals from two sides. The modules use received data to share and compare with all others in a given state. If a *majority* of modules agree on an interpretation of the IR signal, the system as a whole chooses the action corresponding to the majority's decision. In our work, actions are simple commands such as "go forward," "go backward," "stop," "turn," "go limp," etc. Experiments are performed on two configurations: a seven-module caterpillar structure (Fig. 3) and a sixteen-module quadruped structure (Fig. 4).

A majority is needed for a configuration to decide on an action; therefore, up to half of the modules in the system can be

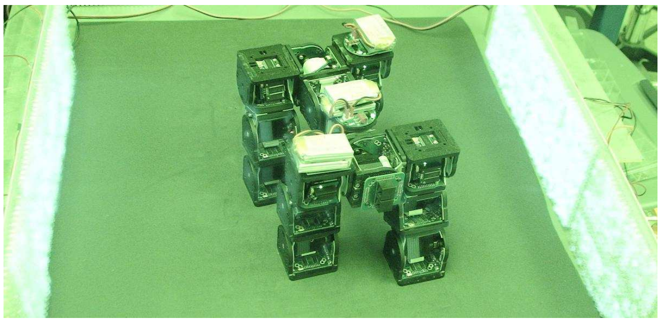

Fig. 4. Sixteen module quadruped receiving IR signals from broadcasting boards. Digital camera in night vision mode captures the illuminated IR LEDS.

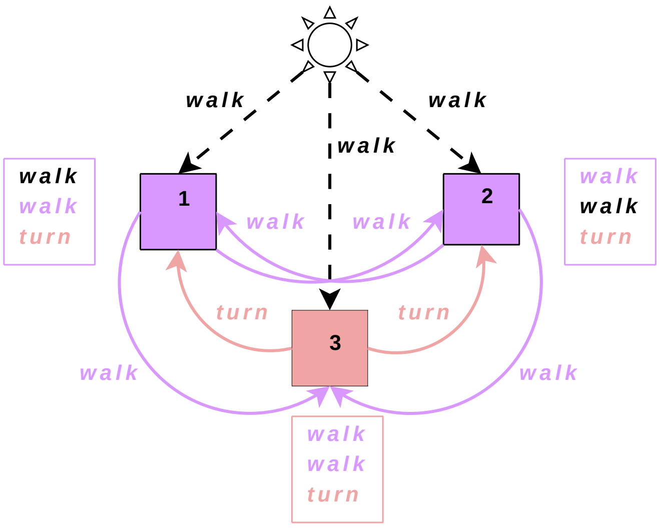

Fig. 5. Three-module schematic of IR signal observation and subsequent sharing of data between the modules. Modules 1 and 2 correctly receive the message as "Walk," where as module 3 incorrectly interprets the message as "Turn." All modules come to the same majority decision to "Walk."

in error (e.g., faulty receivers or dropped CAN messages). In this way, the system is robust and tolerant to certain hardware and software failures during the course of its runtime. Fig. 5 illustrates the basic idea behind this approach. Note that each module ends up with the same list of observed *votes*, including each module's own observations.

From a practical standpoint, we have found this implementation to be useful as we have observed IR signals to be somewhat erratic. Since the IR receivers must convert the analog IR packets to binary pulses according to a certain baud rate, it is common for bytes to be occasionally misinterpreted (e.g., 0b00000110 can be confused with 0b00001110). Together with simple occasional hardware issues (burned out or broken off LEDs), this fault tolerance scheme has demonstrated a level of robustness not possible with only one IR communication path to configurations of modules.

Fig. 6 illustrates the fault tolerance state machine on each CKBot module's processor.

Basic features of the algorithm to note are that:

1) All modules create a list of all other modules in the configuration.
2) Each module continually cycles through all 7 of its IR RX ports, so the sources of data can come from any direction.
3) When an IR signal is received, each module shares this information with all others.
4) All modules record the sightings of all other modules.
5) From the list of sightings, all modules compute a majority.
6) The module with the lowest ID synchronizes the steps for gaits.
7) The system as a whole continues with the decided action ("walk," "turn," "stop," etc.) until a new, different signal is broadcasted and a new majority is computed.

With this algorithm, all CKBot modules in a given configuration have complete knowledge of other modules in the system and also the data that all modules observe from the IR beacons. Each module building the complete list of modules is possible through the logging and ordering of *heartbeats* that all modules broadcast on the CAN. Module heartbeats are CAN messages broadcasted once per second from each module specifying the sending module's ID. Therefore, we allow two seconds after boot up to allow time for at least one heartbeat message from each module to be broadcasted. Each module builds a module configuration list in the span of these two initial seconds.

In a similar manner, the observations (majority votes) are also broadcasted from each module upon IR sightings. When a module receives an IR signal, an interrupt is triggered and the observation is broadcast on the CAN. All modules receive the identical CAN message and store it in a list of ordered module votes.

The CKBot module hardware is organized in manner that allows only one IR port to be accessible at a time. To allow sufficient time for data reception, each CKBot CPU cycles through the seven ports at a rate of 5 Hz. Therefore, to ensure enough time for signals to be received and stored, we program the IR broadcaster to send at a rate equal to double the rate of port switching: 10 Hz. In this way, each of the IR receiver ports is selected for listening in a span of time when at least one IR message is broadcasted.

Because the broadcasters are sending identical messages many times per second, the modules only share their observations if a *new* IR signal is detected. Therefore, the system will carry along its selected task only until a majority of new votes comes in and overrides the old majority decision. In this way, decisions are made on-the-fly and the systems is tolerant and adaptable to data and hardware glitches during the course of its runtime.

Once an action is selected, all modules locally have the same decision and the system is ready to carry out its task as a unit. However, since the times to reach the final decision may be off by up to half of a second, a coordinator module whose sole purpose is to synchronize the time-critical steps of

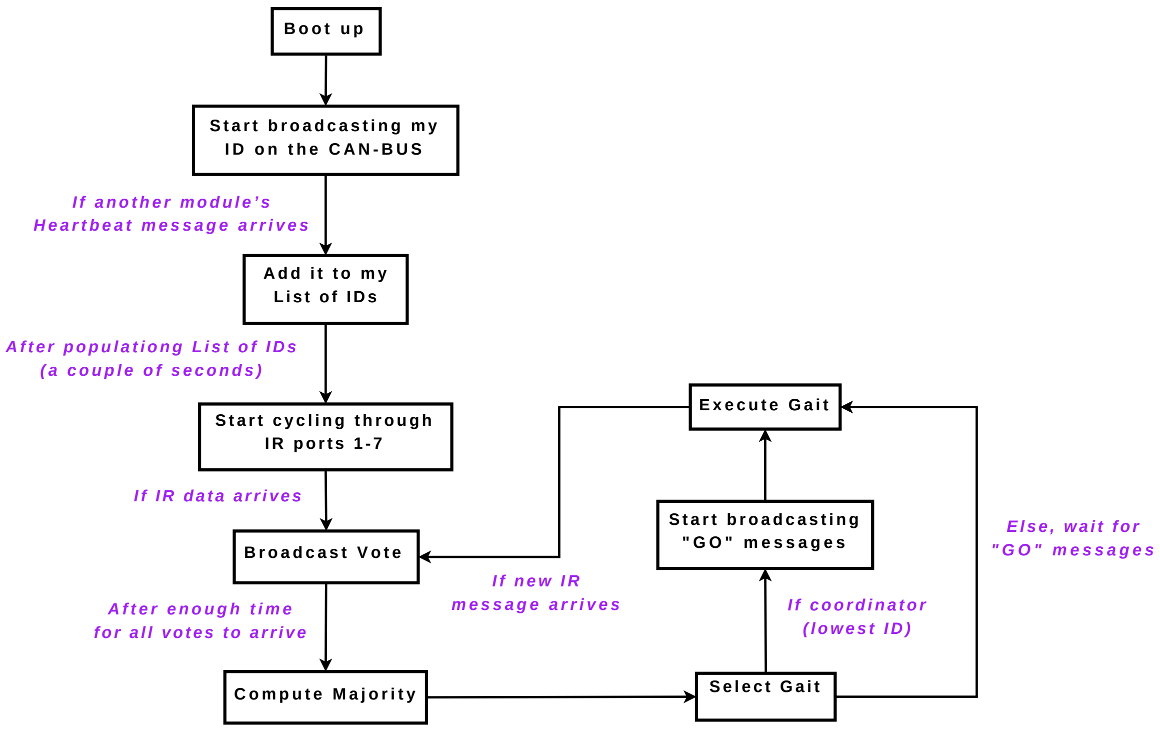
Boot up
Start broadcasting my ID on the CAN-BUS
If another module's Heartbeat message arrives
Add it to my List of IDs
After populationg List of IDs (a couple of seconds)
Start cycling through IR ports 1-7
If IR data arrives
Broadcast Vote
After enough time for all votes to arrive
Compute Majority
Execute Gait
If new IR message arrives
Start broadcasting "GO" messages
If coordinator (lowest ID)
Select Gait
Else, wait for "GO" messages

Fig. 6. Flow chart diagram of the processor state machine on each CKBot module.

a gait (each module's position updates at a rate of 60 Hz) is designated. The coordinator broadcasts "GO" commands on the CAN which all modules (including the coordinator itself) use for well-timed position control. We choose the lowest ID in a given system of modules to be the coordinator; this choice is ad-hoc as the particular ID and module is not significant. If a system's coordinator module were swapped with another, then the next lowest would take over as coordinator and the system would carry on seamlessly. All that is required is that there be one coordinator, and this can be designated at any time.

Modules that simply do not receive any IR signals because they do not have access to the broadcasted signals are handled by the algorithm and are effectively treated as faulty modules. The two "shoulder" torso modules in the quadruped (Fig. 4) are examples of this. All modules that either miss messages or receive erroneous ones are overruled by a majority decision.

The issue of how individual modules know how to behave within the overall structure is determined by each module's position in the group. In particular, the modules are arranged in accordance with their unique IDs. Note that particular node IDs are not significant, rather the *virtual node IDs* that are a simply an ordered mapping (e.g., actual IDs 0x12, 0x15, 0x23 mapped to virtual IDs 0x01, 0x02, 0x03, respectively). Each module maintains a library of gaits in their local program memory and selects individual motion primitives from these gaits depending on their virtual ID. It is possible and may be desirable to extend this approach to be isomorphic so that reorderings of module arrangements do not affect the overall motion of a given fault tolerant system, as studied in an earlier paper [22].

As a side note, since all modules have identical programs, we employed a CAN driven programming scheme which re-programs all modules in a system simultaneously [25]. This is in contrast to the more traditional approach of re-programming each module individually and having a centralized controller have a distinct type of program. The network programming feature greatly facilitated the development of the distributed fault tolerance algorithm.

3.1. System Gait Control

As mentioned earlier, modules select motion primitives from gait tables according to their virtual node ID position within the system. For example, Fig. 7 shows the gait control table for the walking configuration (Fig. 4) to move forward. Each module contains this gait information. In this table, the columns are associated with modules in order of virtual node ID, the rows correspond to gait steps, and the elements in each table correspond to the angle in degrees of the joint for that module in joint space. For instance, if module 0x81 was second in the list of (0x77, 0x81, 0x92, ...) then module 0x81 would select the second column in Fig. 7 as its choice if the majority action was to walk forward. Without this mapping procedure, a module would select the wrong sequence of actions and move inappropriately, even if correct majorities are reached. For example, if module 0x81 in the above quadruped example were to select gaits from column three instead of column two, the robot would fall instead of walk.

List of Ordered Virtual Node IDs

Gait Steps

-28	00	00	00	28	00	00	00	00	-28	00	07	00	07	28	00
-30	00	28	-28	00	00	28	00	00	-30	00	00	-28	00	00	00
-30	00	00	00	00	00	00	00	00	-30	00	00	00	00	00	00
00	00	-28	28	00	00	-28	00	00	00	00	00	28	00	00	00
28	00	-30	00	-28	00	-30	00	00	28	00	05	00	05	-28	00
00	00	-30	00	00	00	-30	00	00	00	00	00	00	00	00	00

Fig. 7. Walking gait control table for the 16 module quadruped. Each module contains this information and and at runtime selects the appropriate column according to its local position within the overall structure.

A zero element in the table corresponds to a module being straight. Once a module knows where it is in the configuration, it uses one column of this table to perform the gait. Should the modules be reconfigured, they would use a different column corresponding to their new position in the configuration.

3.2. Experiments

We implemented the algorithm on the 7 module caterpillar and 16 module quadruped, shown in Figs. 3 and 4. The opposing IR broadcasting boards allowed approximately a two-square-foot range in which the caterpillar could crawl and the quadruped could walk around. We implemented a Zigbee wireless unit to show the status of the modules and monitor the CAN data.

With a hand-held joystick controller, we mapped gait commands to the IR broadcasting boards, which the modules used to decide upon actions. Videos of this work in action can be found online at the ModLab webpage [25].

As expected, the robots were sensitive to orientation with respect to the broadcasting boards. For instance, the caterpillar and quadruped received the signals when parallel to the IR broadcasting boards, with an approximately 15-degree margin of deviation. The turning command was therefore quite useful for the quadruped configuration, as it allowed the robot to stay within the orientation where it could receive the IR commands. Also, when modules in configurations moved beyond a few inches beyond the illuminated area, they stopped reporting signals, as expected.

The IR receivers reported incorrect IR messages 10 percent of the time, on average. The frequency of incorrect observations is function of proximity to the boards, and the particular type of data sent. For instance, 0b11111111 is confused less often than 0b00001111 (which may be confused with 0b00011111 or 0b00000111, if the microprocessor makes a bandwidth miscalculation).

We have constructed a hand held beacon that sends commands which we intend to introduce as "jammers" to intentionally confuse the robots. Experiments on this jamming are now in progress and will be included in future work.

4. Algorithm Design Considerations

An observer of this work may wonder why a majority decision approach is chosen to demonstrate fault tolerance. An alternative would be for a module that fails to receive an IR message to ask its neighbors what they saw. However, this approach does not work in larger systems where communication lines (IR or wireless, if chosen) may be out of scope in patches. That is, if one module in a section within a system asks its neighbors what they saw, the neighbors themselves have not seen any message. It is possible for this message querying to propagate until a module reports an IR message, but then, what if this data is incorrect due to a noisy receiver? In short, we believe our approach to be more general than fault handling through local messaging. With majority certainty, within the entire system, each module makes its decision with high confidence and simplicity.

Also, the line-of-sight requirement for this IR system is specific to demonstrate the algorithm developed. Collective decision-making is, of course, not limited to IR systems. One can readily incorporate the same approach for wireless broadcasting and smart camera systems that are similiarly well-suited for the communication fault tolerance and have been implemented in other works on CKBot.

A benefit of this distributed algorithm is that both computation and number of messages scales with the number of modules in a configuration. Though each modules sends a few messages per decision reached, the number of messages increases only with the number of modules for larger systems. Therefore, we believe this method is quite suitable for large systems.

A cyclic redudancy check (CRC) is a common method to help robustness in communications. It is a method that detects errors in transmission; however, in the case where there is no acknowledgement (as the quadraped control example), there is no means to ask for a resend. Our method acts as an error correction in addition to error detection.

Generalizing this algorithm to modular systems with no IDs is possible if message signatures are assigned. For instance, if modules with no IDs are used to find a majority over a wireless network, the modules might add distinguishing signatures on messages, such as "Leg module," "Foot module," and so on.

Quantifying levels of robustness poses interesting questions. How robust should a system be? What is an optimal level of fault tolerance for a modular robotic system? The percent majority determines the threshold of a system's fault tolerance; choosing this value most likely depends on the tasks and environmental conditions on hand.

In the limiting case where all modules in a configuration are required to see the same signal and 100% agreement is required, there is no effective fault tolerance and the system is as delicate as a centralized controller. Just one faulty IR receiver and the system is paralyzed. However, 100% agreement greatly boosts confidence if a decision of action is fact reached.

In some cases, only one module in a system needs to see a signal, and this may be sufficient. In this other limitng case, there is again no effective fault tolerance, as the system may have numerous conflicting votes from modules and the system is at a loss to determine which one to choose. However, the system has the added advantage that only one module is required to work, which is superior to the requirement that one designated module that must work or else the whole system fails.

So clearly the limiting cases of one or all for majority decisions are not useful for fault tolerance. One interesting majority is the at least 2/3 majority, which guarantees Byzantine fault tolerance [26]. This scenario gives a deeper level of fault tolerance in that the network messages are checked to determine if a subset of modules is intentionally sending confusing data. For example, in a Byzantine system applied to our experiment, all modules would iteratively ask one another what they heard from all the others. If less than one third of the modules contained viruses and were programmed to lie about their IR observations, the non-virus-infected modules would still be able to determine what the original broadcasted message was and choose the correct course of action. In this way, the system would still determine a majority, even in the presence of erroneous shared messages. This approach is quite interesting; however, it is quite computationally and bandwidth intensive as the number of inter-module messages scales as factorial with respect to the number of modules the system. There are, however, practical methods to improve this scaling which may be worth pursuing for modular robotic systems [27].

We are in the process of determining a quantitative measure of confidence in a majority decision. A naïve measure of confidence in a collective decision is simply $C = M/T$, where C is confidence in the majority decision, M is the number of modules in the majority, and T is the total number of modules in the system. However, considering the votes of the modules *not* in the majority may offer information about the confidence in the system's decisions. As an example, a $3/5$ majority with the two dissenting modules giving different votes intuitively suggests a higher level of confidence than a $3/5$ majority with the two dissenting modules giving identical votes. Confidence would be even lower if the two dissenting votes had binary data similar to those for the majority which could be misinterpreted in the signal processing (e.g., 0b00000001 compared with 0b00000011).

5. Conclusion

We have presented a model for modular robotic fault tolerance and demonstrated it on the CKBot system for caterpillar and quadruped configurations. In this approach, we allowed modules to share observations so that they could all vote on actions to take corresponding to the globally broadcasted IR signals. This allows fully functional modules to override erroneous observations of the individuals to select robust actions, tolerant to chronic and intermittent errors in data. We also discuss issues regarding the majority function of the algorithm, and, in particular, rule out the limiting cases of one and all as being effectively fault intolerant systems. We believe this area to be rich with investigations for future work, including quantifying confidence in collective decisions, incorporating simultaneous distinct signals to allow the possibility of multiple decisions, and making the algorithm isomorphic to structural rearrangements.

System consensus will be further tested and developed in future work that will involve multiple-beacon interactions with modular configurations. We intend to introduce hand held IR broadcasters, forcing the modular robot to make decisions in slightly more complex environments where coordinated, robust control is harder to attain. This approach would handle intentionally differing messages, generalizing the algorithm to take into account more decision-making scenarios.

Acknowledgements

Thanks to Robert Callan for his work on the CAN Bootloader, Willy Bernal-Heredia for his work developing the quadruped gait, Matthew Piccoli with his help on GUI debugging and programming, Alex Tozzo and Oscar Nuñez with their help assembling the IR broadcasting boards, Babak Shirmohammadi with his technical help on the IR broadcaster, and Jimmy Sastra for his useful discussions on the general problem.

References

[1] B. Kirby, J. Campbell, B. Aksak, P. Pillai, J. Hoburg, T. Mowry, and S. C. Goldstein, "Catoms: Moving robots without moving parts," in *Proc. of the National Conference on Artificial Intelligence*, vol. 20, no. 4. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 1730.

[2] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund, "Modular atron: modules for a self-reconfigurable robot," in *P IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2004, pp. 2068–2073.

[3] E. Tuci, R. Gross, V. Trianni, F. Mondada, M. Bonani, and M. Dorigo, "Cooperation through self-assembly in multi-robot systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 115–150, 2006.

[4] T. Fukuda and Y. Kawauchi, "Cellular robotic system (cebot) as one of the realization ofself-organizing intelligent universal manipulator," in *P IEEE International Conference on Robotics and Automation (ICRA)*, Cincinnati, OH, USA, May 1990, pp. 662–667.

[5] H. Lipson and J. Pollack, "Towards continuously reconfigurable self-designing robotics," in *P IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2000.

[6] C. Ünsal, H. Kýlýççöte, and P. K. Khosla, "I(ces)-cubes: A modular self-reconfigurable bipartite robotic system," in *P SOC PHOTO-OPT INST Sensor Fusion and Decentralized Control in Robotic Systems II*, 1999, pp. 258–269.

[7] D. Rus and M. Vona, "Self-reconfiguration planning with compressible unit modules," in *P IEEE International Conference on Robotics and Automation (ICRA)*, Detroit, 1999.

[8] J. W. Suh, S. B. Homans, and M. Yim, "Telecubes: mechanical design of a module for self-reconfigurable robotics," in *P IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, 2002.

[9] B. Salemi, W. M. Shen, and P. Will, "Hormone-controlled metamorphic robots," in *P IEEE International Conference on Robotics and Automation (ICRA)*, 2001, pp. 4194–4199.

[10] Z. Butler, K. Kotay, D. Rus, and K. Tomita, "Generic decentralized control for a class of self-reconfigurable robots," in *P IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 2002.

[11] S. Vassilvitskii, M. Yim, and J. Suh, "A complete, local and parallel reconfiguration algorithm for cube style modular robots," in *P IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 2002.

[12] F. H. Bennett and E. G. Rieffel, "Design of decentralized controllers for self-reconfigurable modular robots using genetic programming," in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware (EH-2000)*, 2000.

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

[14] J. W. Burdick, J. Radford, and G. S. Chirikjian, "A 'sidewinding' locomotion gait for hyper-redundant robots," *ADV ROBOTICS*, vol. 9, no. 3, pp. 195–216, 1995.

[15] J. Sastra, S. Chitta, and M. Yim, "Dynamic rolling for a modular loop robot," in *Proc. of International Symposium on Experimental Robotics*, Rio de Janeiro, Brazil, 2006, pp. 421–430.

[16] J. Sastra, W. G. Bernal-Heredia, J. Clark, and M. Yim, "A biologically-inspired dynamic legged locomotion with a modular reconfigurable robot," in *Proc. of DSCC ASME Dynamic Systems and Control Conference*, Ann Arbor, Michigan, USA, October 2008.

[17] K. Støy, W. M. Shen, and P. Will, "Global locomotion from local interaction in self-reconfigurable robots," in *Proc. of the 7th International Conference on Intelligent Autonomous Systems (IAS-7)*, Marina del Rey, California, March 25-27 2002.

[18] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji, "Self-assembly and self-repair method for a distributed mechanicalsystem," *P IEEE International Conference on Robotics and Automation (ICRA)*, vol. 15, no. 6, pp. 1035–1045, 1999.

[19] C. Bererton and P. Khosla, "An analysis of cooperative repair capabilities in a team of robots," in *P IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, 2002.

[20] M. Yim, B. Shirmohammadi, J. Sastra, M. Park, M. Dugan, and C. J. Taylor, "Towards robotic self-reassembly after explosion," in *P IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, October 29-November 2 2007, pp. 2767–2772.

[21] M. D. M. Kutzer, M. Armand, D. H. Scheidt, E. Lin, and G. S. Chirikjian, "Toward cooperative team-diagnosis in multi-robot systems," *INT J ROBOT RES*, vol. 27, no. 9, pp. 1069–1090, September 2008.

[22] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *INT J ROBOT RES*, vol. 27, no. 3-4, pp. 403–421, March/April 2008.

[23] D. Gomez-Ibanez, E. A. Stump, B. P. Grocholsky, V. Kumar, and C. J. Taylor, "The robotics bus: A local communications bus for robots," in *P SOC PHOTO-OPT INST*, ser. Mobile Robot XVII, D. W. Gage, Ed., vol. 5609, Philadelphia, PA, December 2004, pp. 155–163.

[24] (2008) Pcan-view. [Online]. Available: http://www.peak-system.com/

[25] (2009) modlab. [Online]. Available: http://modlab.seas.upenn.edu/wiki

[26] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM T PROGR LANG SYS*, vol. 4, no. 3, pp. 382–401, July 1982.

[27] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proc. of Third Symposium on Operating Systems Design and Implementation*, New Orleans, USA, February 1999, pp. 173–186.