Fall 12-22-2009

Configuration Recognition, Communication Fault Tolerance and Self-reassembly for the CKBot

Michael G. Park
University of Pennsylvania, parkmich@seas.upenn.edu

Configuration Recognition, Communication Fault Tolerance and Self-reassembly for the CKBot

Abstract
We present and experimentally verify novel methods for increasing the generality of control, autonomy and reliability for modular robotic systems. In particular, we demonstrate configuration recognition, distributed communication fault tolerance, and the organization and control of self-reassembly with the Connector Kinetic roBot (CKBot). The primary contribution of this work is the presentation and experimental verification of these innovative methods that are general and applicable to other modular robotic systems. We describe our CKBot system and compare it to other similar, state-of-the-art modular robotic systems. Our description and comparison highlights various design developments, features, and notable achievements of these systems. We present work on isomorphic configuration recognition with CKBot. Here, we utilize basic principles from graph theory to create and implement an algorithm on CKBot that automatically recognizes modular robot configurations. In particular, we describe how comparing graph spectra of configuration matrices can be used to find a permutation matrix that maps a given configuration to a known one. If a configuration is matched to one in a library of stored gaits, a permutation mapping is applied and the corresponding coordinated control for locomotion is executed. An implementation of the matching algorithm with small configurations of CKBot configurations that can be rearranged during runtime is presented. We also present work on a distributed fault-tolerance algorithm used to control CKBot configurations. Here, we use a triple modular redundancy approach for CKBot units to collectively vote on observations and execute commands in the presence of infrared (IR) communication failures. In our implementation, we broadcast infrared signals to modules which collaboratively vote on a majority course of action. Various gait selections for a seven module caterpillar and sixteen module quadruped with faulty subsets of IR receivers have been verified to demonstrate the algorithm's robustness. Lastly, we present work on the communication hierarchy and control state machine for the Self-reassembly After Explosion (SAE) robot. Here, we discuss the interaction and integration of the various sensory inputs and control outputs implemented for camera-guided self-reassembly with CKBot. This section describes the overall communication system and reassembly sequence planning after a group of CKBot clusters is kicked apart.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Mechanical Engineering & Applied Mechanics

First Advisor
Mark Yim

Keywords
modular robotics, reconfigurable robots, configuration recognition, fault tolerance, self-reassembly

CONFIGURATION RECOGNITION, COMMUNICATION FAULT TOLERANCE AND SELF-REASSEMBLY FOR THE CKBOT

Michael G. Park

A DISSERTATION

in

Mechanical Engineering and Applied Mechanics

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2009

Supervisor of Dissertation

Mark Yim
Associate Professor of Mechanical Engineering and Applied Mechanics

Graduate Group Chairperson

Pedro Ponte Castañeda
Professor of Mechanical Engineering and Applied Mechanics

Dissertation Committee

Vijay Kumar, Professor of Mechanical Engineering and Applied Mechanics

George Pappas, Professor of Electrical and Systems Engineering

Mark Yim, Associate Professor of Mechanical Engineering and Applied Mechanics

Acknowledgements

Thanks to my dissertation advisor Prof. Mark Yim for his insightful guidance and steady support. Thanks to the members of ModLab at Penn: Jimmy Sastra, Kevin Galloway, Paul White, Chris Thorne, Bill Mather, Matt Piccoli, Willy Bernal-Heredia, Alex Teichman, and Sachin Chitta. Thanks to Babak Shirmohammadi for his various technical help, Daniel Gomez-Ibañez for his prototypes and early instruction, and Profs. C. J. Taylor and Herbert Wilf for their helpful discussions. Lastly, thanks to my family for their support throughout my time at Penn.

ABSTRACT

CONFIGURATION RECOGNITION, COMMUNICATION FAULT TOLERANCE AND SELF-REASSEMBLY FOR THE CKBOT

Michael G. Park

Mark Yim

We present and experimentally verify novel methods for increasing the generality of control, autonomy and reliability for modular robotic systems. In particular, we demonstrate configuration recognition, distributed communication fault tolerance, and the organization and control of self-reassembly with the Connector Kinetic roBot (CKBot). The primary contribution of this work is the presentation and experimental verification of these innovative methods that are general and applicable to other modular robotic systems. We describe our CKBot system and compare it to other similar, state-of-the-art modular robotic systems. Our description and comparison highlights various design developments, features, and notable achievements of these systems. We present work on isomorphic configuration recognition with CKBot. Here, we utilize basic principles from graph theory to create and implement an algorithm on CKBot that automatically recognizes modular robot configurations. In particular, we describe how comparing graph spectra of configuration matrices can be used to find a permutation matrix that maps a given configuration to a known one. If a configuration is matched to one in a library of stored gaits, a permutation mapping is applied and the corresponding coordinated control for locomotion is executed. An implementation of the matching algorithm with small configurations of CKBot configurations that can be rearranged during runtime is presented. We also present work on a distributed fault-tolerance algorithm used to control CKBot configurations. Here, we use a triple modular redundancy approach for CKBot units to collectively vote on observations and execute commands in the presence of infrared (IR) communication failures. In our implementation, we broadcast infrared signals to modules which collaboratively vote on a majority course of action. Various gait selections for a seven module caterpillar and sixteen module quadruped with faulty subsets of IR receivers have been verified to demonstrate the algorithm's robustness. Lastly, we present work on the communication hierarchy and control state machine for the Self-reassembly After Explosion (SAE) robot. Here, we discuss the interaction and integration of the various sensory inputs and control outputs implemented for camera-guided self-reassembly with CKBot. This section describes the overall communication system and reassembly sequence planning after a group of CKBot clusters is kicked apart.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modular robots are autonomous reconfigurable machines that can change their shape to adapt to new circumstances, recover from damage, or accomplish a variety of tasks ordinarily designated for numerous specialized robots. They can reconfigure their structure to crawl through a narrow passage, roll like a hoop, or form a complex robot with many legs. Since they are made of units that can be rearranged, modular robots are versatile and have the potential to be applied to a wide range of applications. For instance, in conditions where volume for packing is a constraint (such as a mission on a spaceship), it is advantageous to have one set of modular robots execute the various tasks that multiple specialized robots would otherwise carry out. Also, since modular robots can be rearranged quickly, they offer the advantage of rapid robotic solutions in unpredictable, unforeseen conditions, such as an emergency search-and-rescue operation kit.

To date, roughly 20 research groups have developed unique modular robotic systems. Some research milestones that have been achieved include: an introduction of locomotion gaits [51], self-repair [28], self-reconfiguration [29], swarming modular systems [26], self-replication [60], externally actuated systems [50], and self-reassembly [55]. Related research pertaining to work presented here will be reviewed in the introductions of subsequent chapters.

With the introduction of each new system came a variety of hardware innovations including common connection mechanisms [11], multiple sensor integrated systems [53], and mechanical latching mechanisms [30], to name a few. At the Modular Robotics Laboratory (ModLab) at the University of Pennsylvania [25], we have developed the Connector Kinetic roBot (CKBot), which we describe in detail in Chapter 2. In the following section, we compare the CKBot system with other similar, state-of-the-art modular robots.

1.1 Comparison of CKBot to Similar Systems

Numerous modular robots have been created in the past 20 years; CKBot has been in existence for the last 5 of those years. The variety is wide, with some research groups having produced multiple generations of their modular robotic system. In this section, we compare CKBot to related modular robotics systems. This comparison will focus on general features and notable research achievements. More in depth discussions on how certain research aspects relate to the work presented here will be included in separate related works sections after the introductions of each chapter in this dissertation.

The common feature of reconfigurability of repeated mechatronic units unites the different types of modular robotic systems. Besides this general theme, the variety of systems that fall in this category of robots are diverse. Some modules are connected in chains and others in lattice formations. Most modules have internal actuators while some rely on external actuation. Some are self-reconfigurable with inter-module latching mechanisms and some require manual reconfiguration. The majority of modules have embedded processors while some early versions have all central computing done off-board. An inclusive, up-to-date article on the various modular robotic systems can be found online [17]; more focused, in-depth presentations on modular systems, classifications, and research directions can be found in the literature [14], [56], [30].

In some respects, the CKBot system is an adaptable modular robot that overlaps certain subdivisions. It is often connected in chains and loops, but since it has symmetric connections on its four faces, it can also be connected in tree and lattice formations. CKBot is usually manually reconfigurable, but is also capable of self-reassembly with the additional magnetic faces and vision-guided locomotion as used in the SAE robot. All modules have independent means of rotational actuation, but the system is also tolerant to "external actuation" when considering reassembly from unexpected, explosive events, as we will discuss in Chapter 5. Lastly, the computation for the system is flexible with some configurations using fully embedded processing, and some that relegate more intensive routines to an off-board PC.

Table 1.1 compares CKBot to some recent and related modular robotic systems. These systems are all based on cubic structures with rotational degrees-of-freedom. Numerous other novel modular robots not included in this comparison are in the literature: [18], [50], [12], [26], [36], [7], [11]. Our comparison highlights some of the strengths and shortcomings of the latest modular robotics system similar to CKBot.

A major strong point for CKBot is its numerous integrated sensors and peripheral devices. We describe these features in detail in Section 2.1. These add-on items have allowed CKBot to be more versatile than other similar systems. It is also noted for its dynamic locomotion capabilities [41], [40]. A hardware shortcoming is its lack of automated latching mechanism. As such, plans for a latching device is in progress as well as a braking mechanism for added strength in high torque applications.

The primary strength of the M-TRAN systems is its motor-driven inter-module docking mechanism. An upgrade from a permanent magnet and shape memory alloy (SMA) latching/disconnection system, self-reconfiguration and cluster flow of modules is a central part of the group's research. The ATRON robot [30] also features an integrated latching mechanism which is central to its design. From a practical standpoint, one limitation of the M-TRAN robot is its homogeneity. In its current

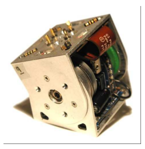
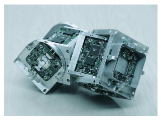
Name	CKBot	M-TRAN	Molecubes	PolyBot	SuperBot
Affiliation	UPenn	AIST Japan	Cornell	PARC	USC ISI
Years Active	2005-Present	1998-Present	2005-Present	1997-2004	2005-Present
Versions	1st Generation	3 Generations Predecessor: Fracta [28]	Successor: Open-source Molecubes [59]	3 Generations Predecessor: Polypod [51]	Predecessor: CONRO [3]
Degrees of Freedom	1 180 ° rotational	2 180 ° rotational	1 120 ° swivel on (111)-plane	1 180 ° rotational	2 180 ° rotational 1 270 ° roll
Docking Mechanisms	manual, optional magnetic faces	motorized latches; magnetic/ SMA release	switchable magnets on faces	G1: manual G2, G3: latches with SMA release	manual
Communi-cation	CAN bus, IR, wireless	CAN bus, IR, wireless	global bus	CAN bus, IR	SPI bus, IR
Features and Devices	accelero-meter, camera, gripper module	accelero-meter, camera	hardware blueprints and software available online	accelero-meter, ratchet brake	unique roll degree-of-freedom
Research Directions	self-reassembly, dynamic locomotion, configura-tion recog-nition	self-reconfigu-ration, CPG-based adaptive locomotion	self-replication	various locomotion, reconfigu-ration planning	distributed hormone control, various locomotion
References	[55]	[29]	[60]	[53]	[37]
Picture					

Table 1.1: Comparison of CKBot with related modular robotic systems.

state, M-TRAN is somewhat limited in the unique tasks it can perform.

Cornell's molecubes feature unique kinematics with their 120° rotational swivel joints on the (111)-plane. This allows for simple "picking-up" motions from "feeder stations" using novel switchable magnets as required for its task of self-replication. One shortcoming of this system is its over-specialization of a modular robot which is perhaps better suited to be a versatile system. Recent developments for an open-source system has encouraged experimentation and development of molecubes for more general applications [59].

The three generations of PolyBot systems have pioneered the integration of electrical components, including IR LEDs, Hall Effect sensors on brushless motors, SMA undocking, accelerometers, and a ratchet brake onto a small modular robot. Common modes of locomotion for a modular robot (rolling, crawling, climbing, etc.) were introduced on this system. PolyBot's latch design for self-reconfigurability is not as reliable as M-TRAN mechanism. It is a dedicated chain-type modular robot whereas CKBot, M-TRAN, and SuperBot are hybrid systems that support both chain and lattice configurations.

USC's SuperBot features the some of latest in the state-of-the art modular robotic hardware. Borrowing the two-cube design from M-TRAN and adding a third "twisting" degree-of-freedom, SuperBot is kinematically less constrained than similar systems. Self-reconfiguration has not yet been achieved with SuperBot, as it currently relies on manual assembly. Though SuperBot is a hybrid modular robot, it has not yet demonstrated effective use of its lattice structural abilities. Computer software for operating systems have had multi-functional and multi-tasking capabilities for decades; robotic hardware has been slower to adopt this level of versatility. Quick change end-effectors and automatic tool-changers were introduced for computer-controlled machining centers in the 1970's, though it has been modular robotics that has pushed for more general progress of application adaptability. Thus, modular robotics research is important for the overall advancement of versatile robots.

1.2 Goals of Modular Robots

Adapting modular robots for highly specialized tasks is a significant challenge. As robots designated for dedicated, unique tasks are often designed from basic principles and task constraints, modular robots are multipurpose and difficult to design with lots of different specialized applications in mind. Also, modular robots may perhaps struggle to achieve the same optimized performance as their specialized robot counterparts, which can be designed with the best possible precision and efficiency *a priori*.

Key goals of modular robotics include scaling down unit module sizes while increasing the number of modules for systems. Trends in these directions would allow modular systems to be increasingly refined and useful for application for various tasks. For advances in smaller scale and greater number, however, autonomous control must be improved as well. In this dissertation, we present work in the areas of modular robot autonomy in an effort toward these goals. Together, these advances would be a significant step toward the "bucket-of-stuff" scenario, where a person could tell a bucket full of randomly strewn modular robots to autonomously self-assemble and do various useful tasks such as "make me dinner" or "change the oil in my car."

A primary feature that will be required for a more autonomous modular robotic system is increased generality of control. A user interacting with each module in a system directly is, of course, undesirable and unnecessary, especially when there can be hundreds of modules, each fitted with individual sensors and communication networks. A better scenario is perhaps one where a modular robot is aware of its resources, environment, and is able to execute desired tasks with a minimum of human guidance and interaction.

Another key goal that will push modular robots toward greater utility is system robustness and fault tolerance. As modular robots rely on increasingly many units, reliability and a method to handle failures in the system become ever more critical.

The biological analogy of metabolic systems that take in environmental resources to repair cellular organisms is an appropriate one for modular robots.

1.3 Dissertation Contributions

In this dissertation, we present and experimentally verify novel methods for increasing the generality of control, autonomy, and reliability for modular robotic systems. In particular, we demonstrate configuration recognition, distributed communication fault tolerance, and the organization and control of self-reassembly with CKBot. These topics encompass some of the central challenges of modular robotics today, including addressing the primary goal of robust, autonomous control for large systems of modular robots composed of many small units. As we will describe, these methods are quite general and applicable to most current modular robotic systems. The key contribution of this work is the presentation and experimental verification of these innovative methods.

After this introduction, we first describe the CKBot system. Our description and comparison will highlight various design developments, features, and notable achievements. The next section deals with isomorphic configuration recognition. In this chapter, we utilize basic principles from graph theory to create and implement an algorithm on CKBot that automatically recognizes modular robot configurations. The third section presents work on a distributed fault-tolerant algorithm used to control CKBot configurations. The last section presents work on the communication hierarchy and control state machine for the Self-reassembly After Explosion (SAE) robot. Each of these sections will begin with a motivation for the problem followed by a background on related research. The sections will also include information on our work to date, and conclude with proposals for future work.

As we will describe in the introduction sections of the subsequent chapters, much of the current state of modular robotic control is highly specialized and tailored for particular applications. These control schemes, while effective for accomplishing specific tasks, are frequently inflexible and sensitive to component failures. In this dissertation, we introduce more general, robust methods for controlling adaptable modular robots.

While the concepts we will present are applicable to various modular robots in existence today, it is hoped that the contributions of this work are also longer-term, with aspects of its approaches built upon and applied to new systems as they arise. Configuration recognition, communication fault-tolerance, and self-reassembly for CKBot captures some of the central problems of modular robots today and it is hoped that the work here has taken modular robotic research one small step closer to an autonomous robot composed of many small modules that can assemble itself and perform tasks from basic, minimal user input.

Chapter 2

CKBot System

In this chapter, we describe the CKBot system. Designed and fabricated in our lab, CKBot is a rotational degree-of-freedom modular robot with various communication and sensing capabilities. It shares some features with other modular robots (as described in the Introduction) and resembles some of its predecessors, notably PolyBot [53] and M-TRAN [29]. The two sections of this chapter describe the hardware and software features of CKBot.

2.1 CKBot Hardware

The basic CKBot module is essentially a cube with an axis of rotation that goes through two opposing cube faces that allows a 180° range of motion. Figure 2.1 shows one module and an assembly of 15 modules that form the self-reassembling robot, as we will describe in Chapter 5. A module is the basic building block of the CKBot system. From these units, various configurations can be built to handle a wide range of applications and modes of locomotion. This design was chosen for its simplicity and general applicability for various locomotion tasks like crawling, rolling, arm-like motions, etc. Each CKBot module is 6 cm x 6 cm x 6 cm and weighs 143 grams.

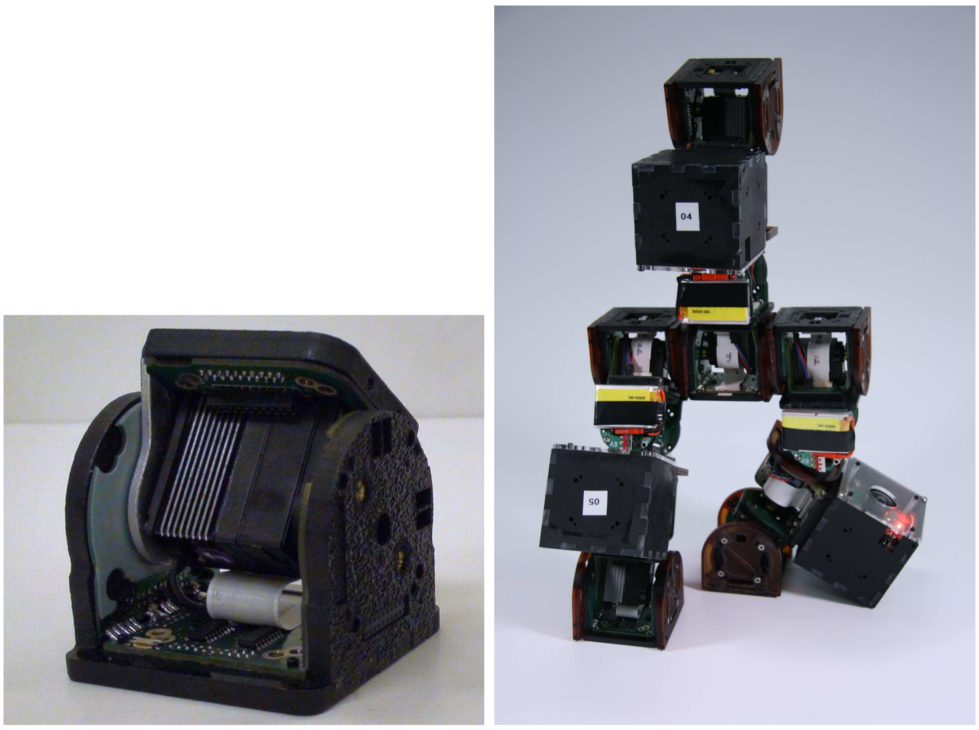


Figure 2.1: One CKBot module and an assembly of 15 modules that form the self-reassembling robot. Photos courtesy Jimmy Sastra.

Each module is equipped with:

- 1 Microchip PIC18F2680 Microcontroller for all module processing,
- 1 Airtronics 94359 Servo for actuating the 180° range of motion,
- 1 Controller Area Network (CAN) transceiver for handling inter-module messaging,
- 7 Infrared (IR) LED Transmitter (TX) and Receiver (RX) Pairs distributed on the four module faces for various IR communication, and
- 8 Identical 20-pin electrical ports distributed on the module for inter-module CAN communication, power distribution, and interfacing to various peripheral devices.

With these basic components, each module is an independent unit that can control its own motions, communicate to other modules on the CAN, and sense connectivity or other data through its IR ports. This design allows for the modules to join and work together intelligently in a myriad of formations. Figure 2.2 shows the basic layout of the primary electrical hardware components. This diagram shows the how each CKBot microcontroller is connected to the CAN bus, the 7 IR ports, and its servo actuator. Note that since the microcontroller has only one pair of serial ports, we use a multiplexer that allows the module to switch between ports with a binary selector. We will describe the software for motor control and communication in the next section of this chapter.

Figure 2.3 shows the layout of the seven IR pairs. These ports are arranged such that the transmitters and receivers of two adjacent modules are aligned for communication. This feature allows for effective neighbor-to-neighbor communication as implemented for configuration recognition and the self-assembly robot (Chapters 3 and 5). The IR receivers are also used for receiving messages from farther distances

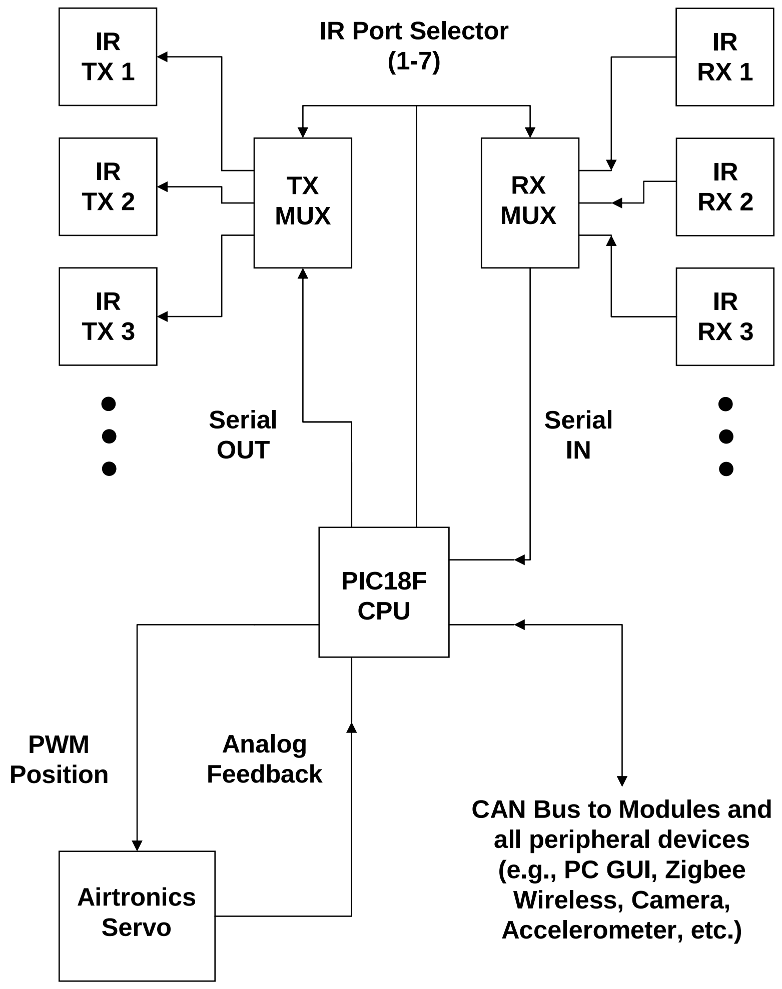


Figure 2.2: Layout of the principal components of the CKBot electrical system.

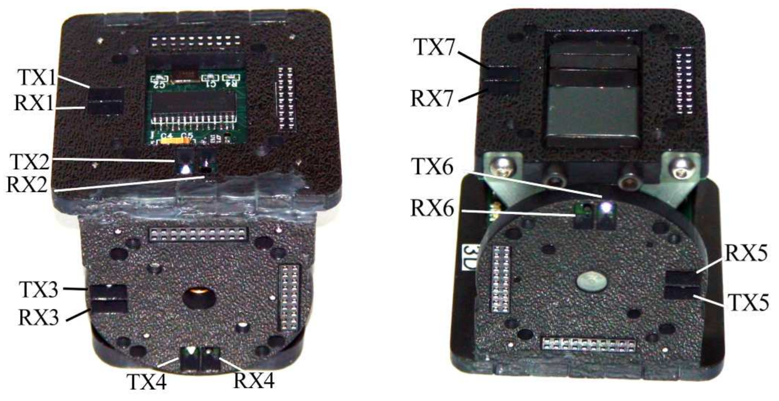


Figure 2.3: Layout of the seven IR transmitter and receiver pairs on the four faces of a CKBot module.

(up to 50 cms away) for the demonstration of communication fault tolerance as we will describe in Chapter 4.

Modules are electrically connected to one another with 20-pin headers that join corresponding sockets between adjacent faces and are held together with screws. The electrical sockets are universal on all module faces and share power (24V, 6V, Ground) and CAN data lines (CAN High, CAN Low). Modules can also connect magnetically to one another with the aid special plates that attach to the faces, as shown in Figure 2.4. These plates screw onto the four CKBot faces to allow modules to quickly connect to one another. Rare-earth permanent magnets are embedded behind the plates and are positioned to join modules in any of the 90° rotational increments that modules connect with electrical headers and screws. Note the mating protrusions and indentations for guiding the connections as well as matching openings for inter-module IR communication. These magnet faces are used in the self-reassembly robot we will describe in Chapter 5.

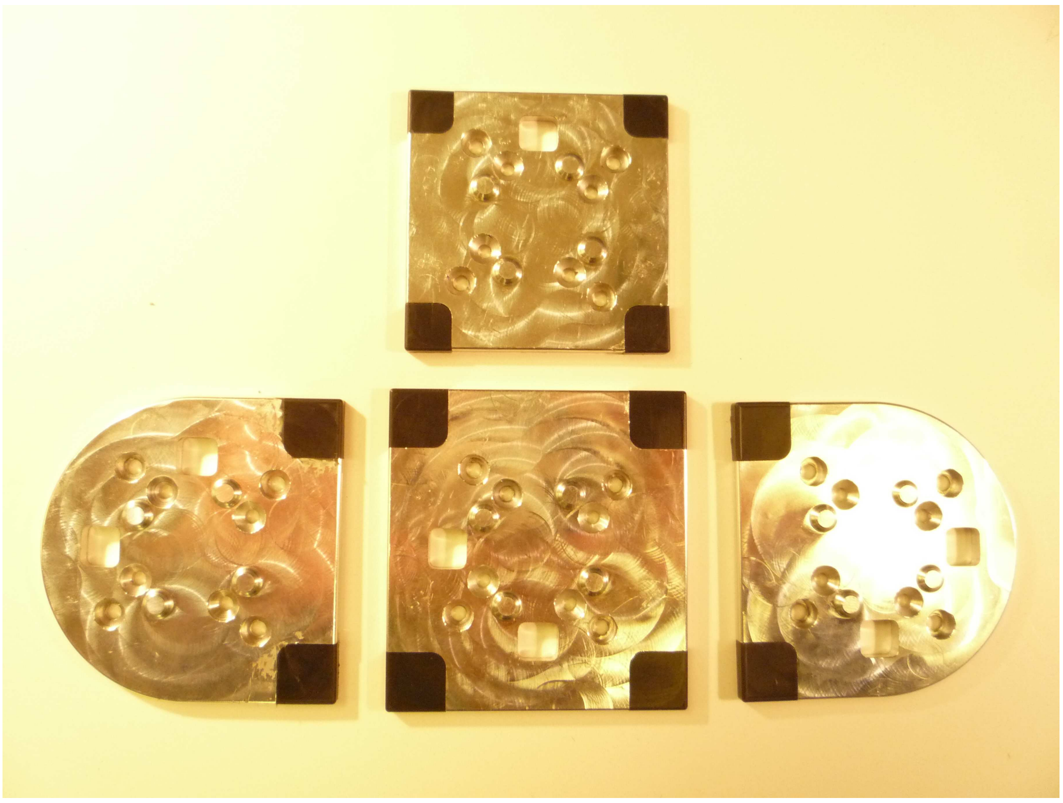


Figure 2.4: CKBot magnetic plates that attach to the outer surfaces of module faces. Rare-earth permanent magnets are embedded behind the plates and are positioned to allow modules to join in 90° rotational increments. Polymer inserts in the corners dampen the fall of modules during the disassembly impact of the self-reassembly robot (Chapter 5).

2.1.1 Peripheral Devices and Other Types of Modules

Over the past five years, the CKBot system has gradually expanded from a simple, one-module type system into a more complex, heterogeneous system with various additional items and modules that add functionality. Some devices are required for most general setups (e.g., batteries, voltage converters) and some are optional and are used for more specialized applications (e.g., ZigBee wireless, magnet faces). Variations of the original module have also been added to the system, chief among them the L7 and Leg Modules. In this section, we will describe most of these additional elements and their applications to the CKBot system.

Table 2.1 summarizes the peripheral devices and other types of CKBot modules. Figure 2.5 shows some of the notable CKBot peripheral devices.

One theme for add-on development is the migration from a tethered CKBot system to a tetherless one. This goal motivated the creation of the lithium-polymer battery board, submodule, and ZigBee wireless chip. The advantages of tetherless control is particularly evident in search-and-rescue type robots where configurations must traverse unpredictable terrain and the SAE robot where clusters of modules begin in random positions and must navigate around each other for reformation.

Another significant motivation for peripheral development is the self-reassembly robot. As we will describe in greater detail in Chapter 5, the SAE robot integrates almost all of the peripheral features of CKBot and, in particular, was the driving force behind the development of the smart camera module, magnet faces, and integrated accelerometer.

The two variations of the original CKBot module, the L7 and Leg Module, were also introduced to address the requirements of certain applications.

The L7 module (Figure 2.5, bottom left) is a structural variation of the standard module which is also called the UBar module (Figure 2.1, left side). As the images show, these modules are given names according to their structural resemblances: the L7 resembles the characters “L” and “7” side-by-side and UBar resembles the letter

Name	Description	Applications
Accelerometer	Orients CKBot configurations with respect to the ground	Self-reassembly After Explosion (SAE) (Chapter 5)
Batteries	Provides power to CKBot configurations that require tetherless control	Rolling Loop [41], SAE, Configuration Recognition (Chapter 3), etc.
DC Power Supply	External power source for tethered CKBot structures	Planetary Contingency [52]
DC-to-DC Converter	Converts 24V to 6V as required by modules	SAE, Dynamic Locomotion, Distributed IR (Chapter 4)
Joystick	Ergonomic control for CKBot structures; crucial for applications requiring teleoperation	Planetary Contingency, Distributed IR
L7 Module	Structural variation of the standard UBar module; useful for axial twisting motions for modules connected in a chain	SAE, Centipede dynamics [40], Arm-like appendage
Leg Module	Higher torque, continuous rotational motion for rolling buggy-type solutions and legged explorers	Planetary Contingency, Search-and-rescue type robots
Magnet Faces	Magnetic inter-module docking for self-assembly during runtime	SAE, Dynamic Planning [47]
PC	User Interfaces that communicate to modules with CAN, ZigBee, IR, FLASH; important for module programming/debugging, robot control, runtime monitoring, gait prototyping	Most CKBot systems
Smart Camera Module	Cameras with distance and angle sensing for module localization and guided locomotion	SAE
Submodule	Stand alone processor that communicates with modules on the CAN for embedded control and IR touch-sensing; CPU fits inside module	Configuration Recognition, SAE, Rolling Loop (using IR touch sensors)
ZigBee wireless	Wireless communication link between modules and PC for controlling and monitoring CKBot structures	SAE, Distributed IR, Centipede dynamics

Table 2.1: Summary of the various peripheral devices and other types of CKBot modules.

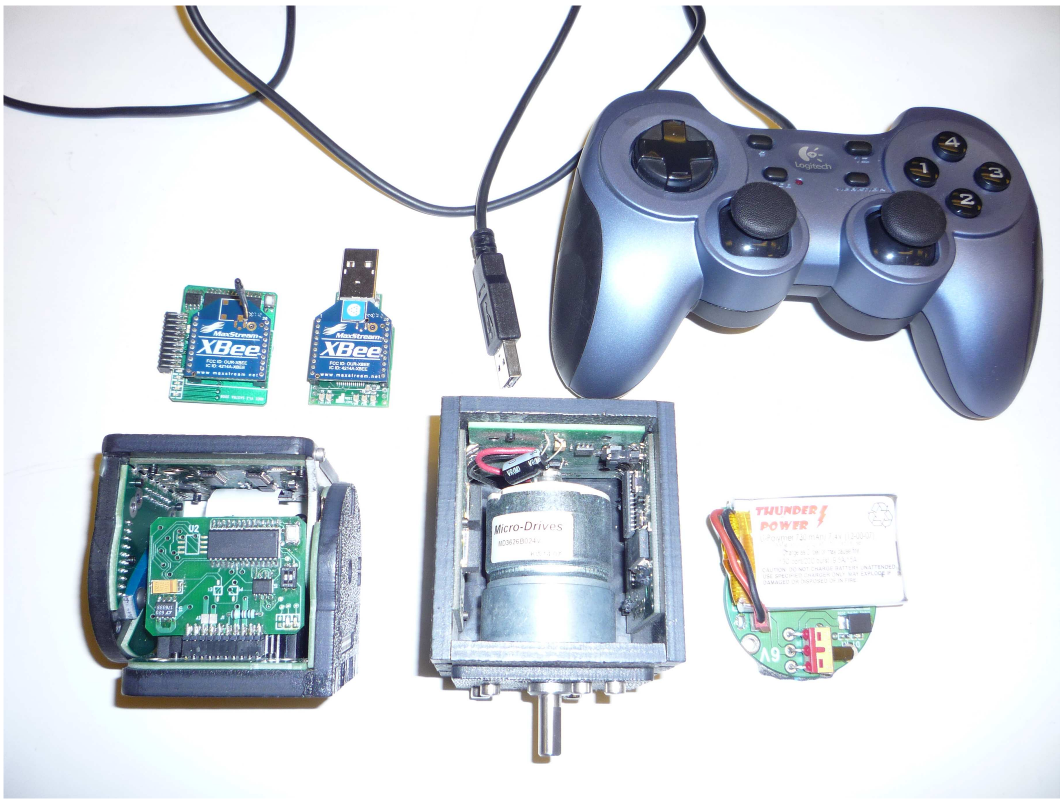


Figure 2.5: Various CKBot peripheral devices. Clockwise from top left: ZigBee wireless system, joystick, Lithium-Polymer battery board, DC Leg Module, and Submodule controller inside an L7 module.

"U" with a bar over it. Both the UBar and L7 modules have identical electronics and software; the only difference between them is structural: U-Bar has three module faces that rotate with respect to the fourth; L7 has two module faces that rotate with respect to the other two. The primary feature of the L7 module is that it allows for twisting, axial motions when connected in chains. This allows for useful degrees-of-freedom for certain tasks like centipede hopping [40] and the self-righting maneuver for the SAE robot as we will describe in Chapter 5. The L7 is the same size and approximate weight as the UBar module.

The Leg Module is equipped with a Micro-Drives MD3626B024V DC motor that is used for powered wheel and leg configurations. Each leg module has the same basic electrical hardware and software of the other CKBot modules. The primary difference is that they allow continuous 360 ° rotation and velocity control. These features make the leg module useful for legged and wheeled CKBot configurations. It uses the same CAN communication protocol as the other modules, but lacks the IR communication capabilities. The inter-module electrical connections are fully compatible with the other CKBot modules. Each leg module weighs 271 grams and is 6 cm x 6 cm x 7 cm making it similar in dimension to the UBar and L7 modules.

2.2 CKBot Basic Software

Each CKBot module is equipped with its own microcontroller, as described in the previous section. While the program memory of modules is occasionally updated and specialized software is sometimes tailored for specific applications, the basic features remain the same. In this section we describe the key elements of the CKBot module's software system. Here we focus our discussion on the standard (UBar) and L7 modules' software.

As Figure 2.2 shows, each module's microcontroller is at the center of its servo actuation and communication to other modules and devices. The 180° servo position range is controlled with pulse-width-modulation (PWM) from the PIC. Analog feedback from the servo to the microcontroller tells the module its actual position: voltage increases linearly from 1.5 V to 2.5 V as servo position increases linearly from -90° to 90°. Each CKBot's microcontroller sends PWM signals at a rate of 60 Hz, the fastest that the servo can handle incoming messages.

Modules communicate with other modules and peripheral devices primarily with CAN and IR. Wireless and camera communication are interfaced through the CAN, so modules rely heavily on this bus messaging system. The CAN protocol is based on the GRASP lab's Robotics Bus architecture [13]. CAN messages transmit data at 250,000 bits per second. IR messages are only a byte long and transmit data at 2400 bits per second. This slower serial rate was chosen to increase IR reliability, as we will describe in more detail in Section 4.4.

All modules and devices on a connected CAN bus have unique node IDs that allow them to communicate with one another and know the sources of these messages. Since CAN messages contain information about the sender, receivers can filter which messages to accept for processing, and which to ignore.

One important periodic message that all modules are programmed broadcast are Heartbeat messages. Heartbeats are CAN messages broadcasted once per second from each module specifying the sending module's ID and are designed to allow modules and users to know which modules are on the bus at all times. This allows all modules, devices, and user interfaces to observe if new modules are added or removed from the system during runtime. We use this feature to allow the configuration recognition and distributed IR systems to automatically detect which modules are in a configuration at any given time.

The CKBot module firmware is written in a C-language and compiled in the Microchip MPLAB Development suite. We use a special GUI developed for CKBot for system tasks such as locomotion development, debugging, and teleoperated control.

The latest extensive information on this programming system and user interface for CKBot is available on the ModLab webpage [25].

Chapter 3

Isomorphic Configuration Recognition

The advantage of reconfiguration is central to modular robotic systems. With this benefit, however, comes a complex and interesting challenge: how does a modular robot recognize which shapes are useful or familiar? The ability for a modular robot to determine which configurations are needed for various tasks is a fundamental requirement for increased autonomy. For example, if a modular robot forms into a snake-like configuration, it should recognize its current state and select the correct corresponding slithering motions. This feature of self-discovery in a modular robot has been proposed (as we will describe in Section 3.2), but the methods to-date are limited in scope or have only been outlined without experimental or simulation verification. The implementation of automatic configuration recognition in CKBot we will present here is general and applicable to most modular robotic systems. We will mention applicability to specific systems in the related work section (Section 3.2).

3.1 The CKBot Configuration Recognition Problem

In the following work, we propose a general approach to solving the configuration recognition problem for modular robots. Our work is distinguished from previous work in that we describe our system from the perspective of a modular robot recognizing its own shape (rather than a program giving instructions on how to construct a robot given a set of parts), we introduce a new, succinct matrix representation of modular robots, we implement our algorithm on our CKBot system, and we show the generality of the approach and how it extends beyond simple configuration-dependent gaits. Furthermore, in showing our approach, we discuss how it can be readily extended to other modular robotic systems.

The works of [35] and [39] are advantageous in that they distribute the recognition problem amongst the modules in a configuration. However, since these approaches rely solely on neighbor-to-neighbor IR communication relaying, the methods occasionally failed through dropped packets and were relatively bandwidth intensive. In our work, we exploit both the global CAN and local IR to solve the configuration recognition problem. From a top-down view, we employ a standalone sub-module chip 3.1 to orchestrate and calculate the configuration recognition algorithm sequence. Therefore, a single sub-module chip (per configuration) acts as a central controller to the system as a whole.

Since the sub-module chip acts as the coordinator and central processor for our system (with each modules' own processors controlling the low level communication, sensing and actuation routines independently), we frame our work from this perspective. Therefore, the first sensible step for the sub-module is to gather inter-module connectivity data and organize it in an organized, useful manner. As mentioned earlier, each CKBot module has a unique node identification (ID) number. Furthermore, each module has 7 IR ports that can be used to determine how modules are

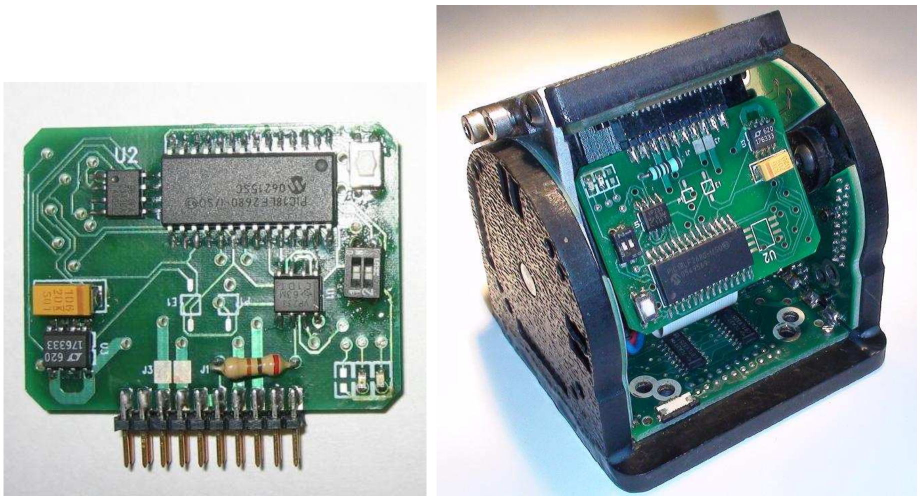


Figure 3.1: A CKBot sub-module controller alone and inside a module.

connected to one another. Therefore, it is a matter of sensing neighbors and building the complete topology of the system which the controller can use.

3.2 Related Previous Work

A few researchers have studied aspects of configuration recognition and proposed solutions. Chen and Burdick [6] showed a method for enumerating the unique (non-isomorphic) configurations of a modular system, given sets of module and connection types. The algorithm is based on using symmetry groups to count only those states that are unique under symmetric rotations or translations. A primary contribution of this work is the introduction of the use of *incidence matrices* to describe modular robot configurations. This algorithm is essentially the inverse of the configuration recognition problem: it gives all possible (unique) constructions of a modular robot given a set of modules, whereas the configuration recognition problem is concerned with a modular robot discovering its own morphology. However, the isomorphic recognition algorithm we will present here also identifies identical structures and unveils structural symmetries associated with the automorphic group for modular robots. A key advantage of our method over Chen and Burdick's is that our algorithm requires no modular input information into the system for configuration recognition, whereas their algorithm requires a definition of the various types of modules and port connections to create the list of unique robots.

Castano and Will [4] proposed a method for configuration representation and recognition of CONRO robots. Using a modified Adjacency Matrix, this work shows the connection from physical robot, to graph representation, to binary matrix form, usable for PC or embedded processors. The paper shows simple method for creating a configuration adjacency matrix and concludes with a suggestion as to how a comparison between the created matrix and stored "catalog" matrix can determine a match. The primary contribution of this work is the presentation of a unique way to represent modular robots; however, no actual configuration recognition implementations were conducted or proposed. Our configuration recognition method uses a distinct type of robot representation and completes essential task of graph matching of isomorphic structures. The method we will present is directly applicable to the CONRO robot, as both CKBot and CONRO have global CAN and local (IR) communication networks necessary for arbitrary configurations to detect their shape.

Butler et al. [1] proposed a method for determining of their 2D Crystalline modular robot is in a goal configuration. This approach uses relay message passing around the perimeter of a Crystalline robot to determine if the shape is the goal configuration decided upon, *a priori*. This method is useful in its simple implementation and direct application to a modular robotic system. However, it primarily deals with solid connected configurations and is not applicable to loop structures or lattice configurations with vacancies. The configuration recognition method we will present in this chapter has the benefit of being general and applicable to a wider variety of shapes. Since each Crystalline module is equipped with only neighbor-to-neighbor

IR communication, our approach is applicable to the Crystalline robot if the data acquisition and configuration detection stages of the algorithm are adapted to support local communication only (signed message propagation, distributed computation of graph matching, etc.).

Stoy, Shen, and Will [46] showed how a CONRO modular robot can change its mode of locomotion based on its configuration. This method is based on modules specifically programmed to send and receive IR packets to determine if a configuration is correct for a particular gait. This method is novel in its application of configuration to gait; however, it is specific in its mapping of gaits based on predetermined shapes. The approach we will present here is more general and considers arbitrary configurations for discovering the shape of a modular robot. As mentioned earlier, our method is directly applicable to the CONRO robot.

Park, Chitta, Teichman, and Yim [31] propose three methods for solving the configuration recognition problem. In this work, some basic relations to graph theory are discussed and the methods for configuration recognition are simulated and compared for performance and applicability to CKBot. The methods presented are quite general and applicable to a wide variety of reconfigurable systems. In particular, the approach by Chitta builds upon an open-source graph automorphism software called **nauty** (**n**o **aut**omorphisms **y**es? [24]) that maps a given configuration matrix for CKBot to the automorphism group of a canonical representation of a known configuration matrix. If the given matrix maps to a known configuration, then a configuration match is found and the corresponding mapping is also determined. This approach has the advantage of more efficiently finding configuration matches; however, the algorithm is too large to run on-board most modular robots. The approach by Teichman uses a variety of heuristic filters to determine if a given configuration is in a catalog of known configurations (such as number of modules, number of connections per module, physical center of mass of the system). The algorithm then chooses a particular module and, in a module-by-module sequence, compares the structure of the given system with the stored configurations in the library. During the sequence, if ever a module within the structure is different from all of the library of configurations, then the configuration is determined to be unknown. If a configuration matches a unique catalog configuration throughout the sequence, then a match and corresponding mapping is found. Teichman's method is relatively quick and uses a simple, low program memory algorithm; however, since it requires a three-dimensional linked list to discover configurations, it requires significant amounts of memory (on the order of gigabytes). The approach by Park utilizes the determinant of the square adjacency matrix to compare the structure of a given configuration with those in the catalog of configurations. In this way, a configuration match can readily be found. The corresponding eigenvector matrix is then used to determine the permutation matrix, and hence the identification (ID) mapping, between the given state and the match found in the library. This algorithm was implemented on board the CKBot system and was shown to be effective for small configurations of modules. The last method is the springboard for this section of the dissertation. Our method of performing graph spectrum computations on incidence matrices is quite general; applicable systems are mentioned in Section 3.2.

3.3 Algorithm Overview

After the submodule controller has acquired all of the port connections for all the modules in a configuration and organized it into the port adjacency matrix, it is ready to perform analysis to determine if the detected configuration is in its catalog of stored configurations. For this work, the primary subsequent steps for configuration recognition are graph matching and graph mapping (finding the permutation between detected and known states). The graph matching step is essentially a comparison of eigenvalues between given and known states, as we will describe in Section 3.6. Once a potential configuration match has been found, the mapping step confirms the match and produces the precise label ordering for detected and known states to be the same. This is done by finding the permutation matrix that reorders the module ID labels, as we will describe in Section 3.7. An example of the matching and mapping procedures for configuration recognition will be presented in Section 3.8.

After a successful configuration detection, the determined label mapping can be used to for isomorphic gait control, feedback, and/or control processing. In essence, the configuration mapping allows the controller to recognize that the given configuration is recognized and any control or feedback loop that is known for an isomorphic permutation of the structure is the same as the given structure, with the module ID permutation mapping as the key for how the detected and known configurations relate.

3.4 Configuration Data Acquisition

To acquire the configuration topology, the sub-module controller instructs the modules to talk and listen in sequential order so that all connections are discovered. Fig. 3.2 shows a schematic of a step in the process of module connectivity data acquisition. In the step in this figure, Module 2 is instructed by the controller to "listen" through its IR ports while Modules 1 and 3 are instructed to transmit their IDs through their IR ports. The modules all take turns listening and talking through all ports so that all possible connections are detected and recorded. Algorithm 1 shows the corresponding pseudo code for how the controller sub-module acquires information about all the connections in the system. Note that the algorithm is a double loop that tells the modules (via CAN) to wait and listen for a signal or to blink their IR signals and talk to all modules. When talking, the modules simply blink their own IDs in serial binary form, on all 7 ports. When listening, each of the 7 ports is selected one at a time, to ensure that the signal is received. After all modules have shared their data, they each report $1x7$ array of the sighting on the ports to the controller, which organizes all the sightings into a matrix.

Algorithm 1 Controller coordinated module connectivity connection sequence.

```
for $i = 1$ to $AllModules$ do
  for $j = 1$ to $AllModules$ do
    if $j \neq i$ then
      Send CAN message: "Talk on all Ports"
    Send CAN message: "Listen on all Ports"
for $j = 1$ to $AllModules$ do
  Send CAN message: "Give me all of your neighbor data"
```

Combinations of port elements account for three additional port numbers, which we define as follows: Ports 8 is when a module observes another through *both* ports 1 and 2, Port 9 is observation through both ports 3 and 4, and Port 10 is observation through both ports 5 and 6. These connections are possible when adjacent faces between modules have *both* IR transmitters and receivers lined up (Fig. 2.3). From an algorithmic standpoint, these connections are no different from the other 7 types of connections. Note that if a module sees another on ports 8, 9, or 10, then the reciprocal module must also have an 8, 9, or 10 port connection.

3.5 The Port Adjacency Matrix

After the sub-module controller receives all neighbor sighting from all modules in the configuration, the first logical step is to organize the data into a useful and succinct form. Based on the usual adjacency matrix used in graph theory [15], we introduce a port-adjacency matrix which describes both the connectivity of modules as well as how modules are oriented in each connection. Like the standard adjacency matrix, the port-adjacency matrix also is an nxn square matrix for n connected modules where nonzero entries denote connections. The diagonal entries are all zero (since we define the modules to not be connected to themselves). The key difference between the standard adjacency matrix and the port-adjacency matrix is that the

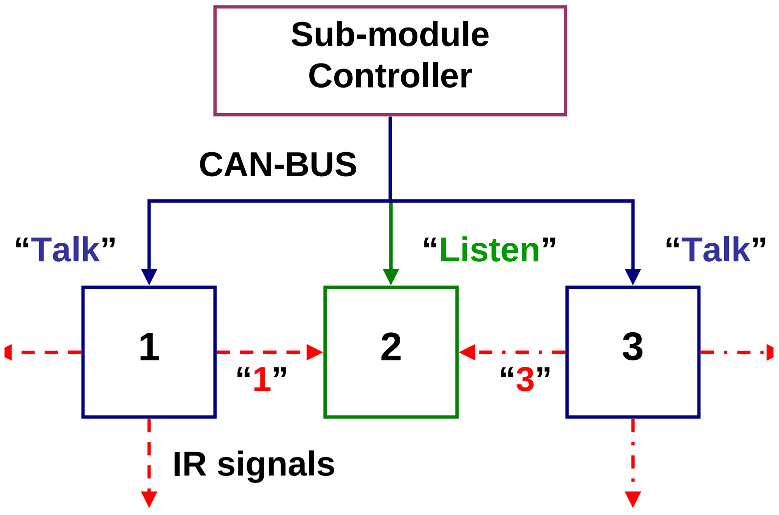


Figure 3.2: Configuration connectivity acquisition schematic for three modules. In this step of the sequence, Module 2 is receiving IR signals from Modules 1 and 3.

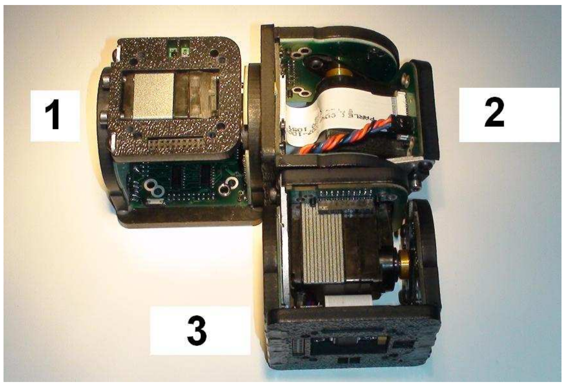
adjacency matrix elements are all either 1 or 0 and, therefore, symmetric, whereas the port-adjacency matrix can have nonzero port elements $1-10$ to denote types of connections between modules and is generally non-symmetric. The port-adjacency matrix obviously reduces to the standard adjacency matrix if orientations are not considered (relevant for perhaps other types of simpler modular robots) and only connectivity is considered.

The port-adjacency matrix is best described with an example. Consider the configuration shown in 3.3. The corresponding port-adjacency matrix is shown to its right. The numbers outside the matrix correspond to the given module IDs 1, 2, and 3. The choice of IDs is, of course, arbitrary; the important thing is that the rows and columns are ordered the same way. The row indicates a receiving module's connectivity to other modules in the system. The 0s indicate no connection (modules are not connected to themselves, by definition), and all nonzero elements denote connections to other modules. The number corresponds to the type of inter-module connection. Note that one module does not necessarily observe a connection to the reciprocal module through the same port.

In particular, Module 1 sees Module 2 through Port 9; Module 2 sees two modules: Module 1 through Port 8 and Module 3 through Port 5; and Module 3 sees Module 2 through Port 7 (refer to Fig. 2.3). In this way, the port adjacency matrix contains a complete description of the topology of a given CKBot configuration and is what we use as a basis for configuration recognition analysis.

3.6 Spectral Graph Analysis of the Port Adjacency Matrix

At this point in this dissertation we digress from the physical implementation of the configuration recognition algorithm and discuss the basis of the method in more general terms and present some of the analysis we conduct in Matlab. We will come



$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 9 & 0 \\ 2 & 8 & 0 & 5 \\ 3 & 0 & 7 & 0 \end{array}$$

Figure 3.3: Example configuration and corresponding port adjacency matrix.

back to the physical implementation after this section.

The method we use for determining configuration isomorphism is based on basic concepts from spectral graph theory. A known method for checking for isomorphism between two graphs is through adjacency matrix spectral decomposition [8]. That is, if two graphs G_1 and G_2 are isomorphic, then the eigenvalues of the corresponding port-adjacency matrices A_1 and A_2 are equal. The inverse, however, is not always true: port-adjacency matrices with identical arrays of eigenvalues are not necessarily isomorphic. To deal with this issue, permutations of eigenvector elements are employed as a confirmation of isomorphism and as a basis for finding the permutation mapping of module IDs.

The other methods studied in conjunction with this work (**nauty**-based and linked-list approaches in [31]) use heuristics to search for matches between two graphs. This approach differs in its use of well-established ideas in spectral graph theory and its applicability to generate approximate methods [58] where other techniques may fail.

Cospectral graphs such as the pair shown in Figure 3.4 [16] are rare and interesting cases that may arise in modular robotic configurations. The characteristic

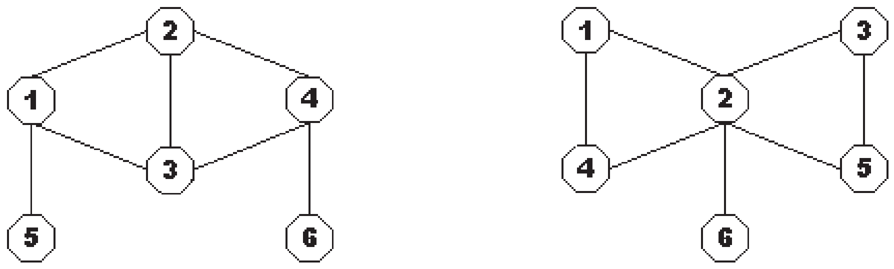


Figure 3.4: Example of cospectral graphs with the same characteristic polynomial: $-1 + 4\lambda + 7\lambda^2 - 4\lambda^3 - 7\lambda^4 + 6\lambda^6$.

polynomials for these graphs are the same despite non-isomorphism. In these scenarios, although the eigenvalues are the same, the structures are not isomorphic since no relabeling of nodes maps one configuration to the other. A comparison of the eigenvectors for these graphs is required to find that permutation does not exist and confirm non-isomorphism.

3.7 Linear Algebra of Adjacency and Permutation Matrices

Given two port-adjacency matrices A_1 and A_2 with the same graph spectrum, we wish to find the *permutation matrix* P that reorders the rows and columns of A_1 so that they are identical to those of A_2:

$$A_2 = PA_1P^{-1}. \tag{3.1}$$

Note that P swaps the rows and P^{-1} swaps the columns of A_1 so that gaits for A_1 can be mapped onto corresponding gaits for A_2. The permutation matrix is composed of only one 1 across any row and column with the remaining entries as

0's. This gives the property that all permutation matrices are orthogonal, satisfying $P^T = P^{-1}$. The identity matrix is a permutation matrix that maps a configuration onto itself.

If A_1 and A_2 are decomposed into their spectral canonical forms

$$\begin{aligned} A_1 &= Q_1 \Lambda Q_1^{-1} \\ A_2 &= Q_2 \Lambda Q_2^{-1} \end{aligned}$$

where Λ is the diagonal eigenvalues matrix (note that they are the same for both since A_1 and A_2 correspond to isomorphic graphs as the rows and columns just interchanged) and Q_1 and Q_2 are the associated eigenvector matrices. This gives

$$\begin{aligned} Q_2 \Lambda Q_2^{-1} &= P Q_1 \Lambda Q_1^{-1} P^{-1} \\ &= (P Q_1) \Lambda (P Q_1)^{-1} \end{aligned}$$

which reduces to

$$Q_2 = P Q_1. \tag{3.2}$$

This shows that the permutation matrix also relates to eigenvector elements of port-adjacency matrices of isomorphic configurations. Therefore, by matching appropriate matrix elements in Q_2 to corresponding elements in Q_1, we can determine the permutation matrix that satisfies Equation 3.1. In the following section, we illustrate this procedure with an example.

3.8 Example of Finding the Permutation Matrix

Consider configuration FL from Figure 3.5. Recall that the port-adjacency matrix corresponding to this configuration is given by:

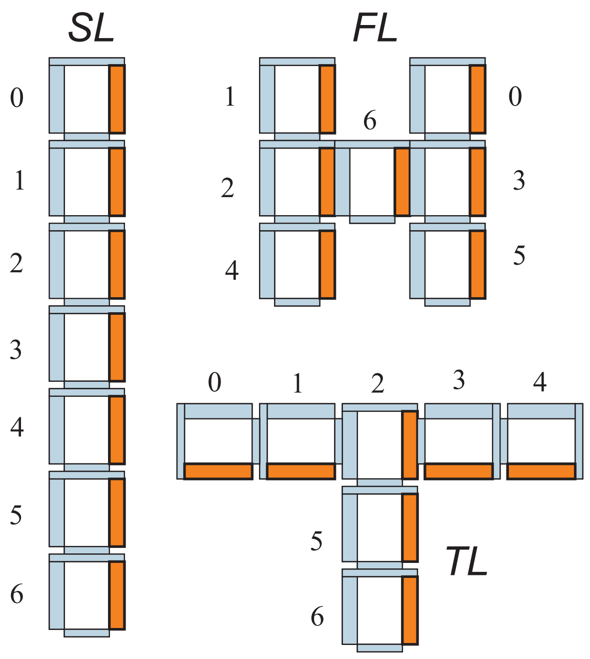


Figure 3.5: A possible database of known configurations.

$$A_{FL_c} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 6 \\ 1 & 0 & 0 & 7 & 6 & 0 & 0 \\ 0 & 1 & 7 & 0 & 4 & 0 & 0 \end{bmatrix}$$

Now, consider another configuration with a port-adjacency matrix:

$$A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 1 & 7 & 0 & 4 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 7 & 1 & 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 6 & 0 & 0 & 4 & 0 \end{bmatrix}$$

First, we note that A_{FL_c} and A_2 have the same characteristic polynomial:

$$\text{Det}(A_{FL_c} - \lambda I) = \text{Det}(A_2 - \lambda I) = \lambda^7 - 76\lambda^5 + 868\lambda^3.$$

This property suggests that these configurations are likely candidates for being isomorphic. To confirm this suggestion (and rule out that these structures are cospectral), we proceed further to find a permutation matrix that satisfies the property in Equation 3.2. We first compute the eigenvector matrices with columns ordered according to the roots of the characteristic polynomial (eigenvalue graph spectrum):

$$\lambda = \begin{bmatrix} 7.87 & 3.74 & -3.74 & -7.87 & 0 & 0 & 0 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} -0.47 & -0.72 & 0.72 & -0.47 & -0.97 & 0.32 & -0.58 \\ -0.31 & 0.48 & -0.48 & -0.31 & -0.18 & -0.93 & 0.12 \\ -0.04 & 0.06 & -0.69 & -0.04 & -0.02 & 0.11 & -0.35 \\ -0.06 & -0.10 & 0.10 & -0.06 & 0.06 & -0.07 & -0.42 \\ -0.53 & 0 & 0 & -0.53 & 0.09 & 0.02 & 0.58 \\ -0.52 & -0.38 & 0.38 & 0.52 & 0 & 0 & 0 \\ -0.34 & -0.25 & 0.25 & 0.34 & 0 & 0 & 0 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} -0.06 & -0.10 & 0.10 & -0.06 & 0.06 & -0.07 & -0.42 \\ -0.47 & 0.72 & -0.72 & 0.47 & -0.91 & -0.83 & 0.83 \\ -0.34 & -0.25 & -0.25 & -0.34 & 0 & 0 & 0 \\ -0.31 & -0.48 & 0.48 & 0.31 & 0.38 & 0.41+0.11i & 0.41+0.11i \\ -0.04 & -0.06 & 0.06 & 0.04 & -0.01 & -0.19+0.04i & -0.19+0.04i \\ -0.52 & 0.38 & 0.38 & -0.52 & 0 & 0 & 0 \\ -0.53 & 0 & 0 & 0.53 & -0.078 & 0.23-0.10i & 0.23-0.10i \end{bmatrix}$$

Note that the columns of Q_1 and Q_2 (the eigenvectors of A_{FL_c} and A_2) are both ordered so that they correspond to the same eigenvalue elements. The columns have been normalized so that the sum of the squares down any column equals one. Also, note that the absolute value of each eigenvalue element is of interest, since eigenvectors, as a whole, can be scaled by a minus sign (vector pointing in the opposite direction) with the eigenvalues and similarity properties of the matrix unchanged. To create the permutation matrix, note that Q_{2ij} (the element in the i^{th} row and the j^{th} column of Q_2) can be written as:

$$Q_{2ij} = P_{i1}Q_{11j} + P_{i2}Q_{12j} + \cdots + P_{i7}Q_{17j}.$$

The property that P has a single 1 across any column or row (with all other elements zero) allows us to build P simply by comparing the permutation of elements down corresponding columns in Q_1 and Q_2.

For instance, we see that $|Q_{111}| = |Q_{221}| = 0.47$. Consequently, $P_{21} = 1$ with all other elements in the rank and file of P_{21} equal to zero. Next, observe that $|Q_{121}| = |Q_{241}| = 0.31$. This gives us $P_{42} = 1$ with all other elements in the rank and file of P_{42} equal to 0. Similarly, $|Q_{131}| = |Q_{251}| = 0.04$ giving $P_{53} = 1$ with all other elements in the rank and file of P_{53} equal to 0. Continuing down the first columns of Q_1 and Q_2, we can construct the following P that confirms isomorphism and gives the desired module labels:

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The mapping is given by:

$$\pi_{1\to 2} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 4 & 5 & 1 & 7 & 6 & 3 \end{pmatrix}$$

Generally, for a column k, when $|Q_{1jk}| = |Q_{2ik}|$, $P_{ij} = 1$ with all other elements across row i and down column j zero. The relabeled graph is shown in Figure 3.6.

3.9 Graph Symmetry and Configuration Mapping

It is evident that the choice of eigenvector for comparison is important. In the above example, if we had chosen any of the eigenvectors associated with the degenerate eigenvalue (zero), we would not have been able to build the permutation matrix. This redundancy in eigenvalues can be attributed to an algebraic regularity in the graph structure [43].

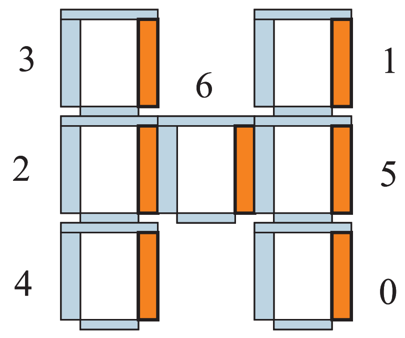


Figure 3.6: Relabeling of A_{FL_c}.

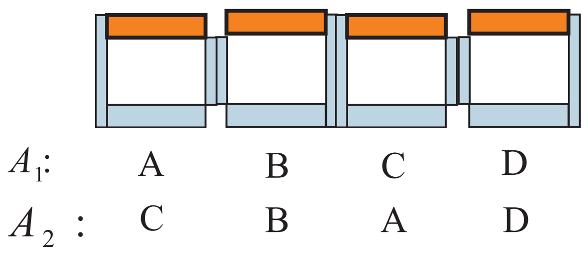


Figure 3.7: Example of symmetric configuration with two permutation matrices that relabel modules in the same way.

Structural symmetry creates interesting scenarios for this method to find the graph isomorphism mapping. Consider the following configurations in Figure 3.7.

The two rows of labels give the adjacency matrices

$$A_1 = \begin{bmatrix} 0 & 7 & 0 & 0 \\ 7 & 0 & 1 & 0 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 7 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 0 & 1 & 0 & 7 \\ 1 & 0 & 7 & 0 \\ 0 & 7 & 0 & 0 \\ 7 & 0 & 0 & 0 \end{bmatrix}$$

with the following eigensystem:

$$\lambda = \begin{bmatrix} -7.52 & -6.52 & 6.52 & 7.52 \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} -0.48 & 0.52 & -0.52 & 0.48 \\ 0.52 & -0.48 & -0.48 & 0.52 \\ -0.52 & -0.48 & 0.48 & 0.52 \\ 0.48 & 0.52 & 0.52 & 0.48 \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} 0.52 & -0.48 & 0.48 & -0.52 \\ -0.52 & -0.48 & -0.48 & -0.52 \\ 0.48 & 0.52 & -0.52 & -0.48 \\ -0.48 & 0.52 & 0.52 & -0.48 \end{bmatrix}$$

Following the same approach as above, we end up with the following permutation matrix:

$$P^* = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

The multiple 1's across the rows and columns occur because of the redundant elements in the eigenvectors. The reason this occurs is because two distinct permutation matrices both satisfy Equation 3.1, namely:

$$A_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 7 & 0 & 0 \\ 7 & 0 & 1 & 0 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 7 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 7 & 0 & 0 \\ 7 & 0 & 1 & 0 \\ 0 & 1 & 0 & 7 \\ 0 & 0 & 7 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

where P^* is the union of P_1 and P_2 given by:

$$P_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The reason two permutations occur for this configuration (and any isomorphic labeling of this graph), is because both matrices are in the graph's symmetry group. Also called automorphic group, we see that the symmetry is a 180^o rotation about the center of mass of the system. For the purpose of spectral decomposition approach, the algorithm tries valid combinations of P^* type unions (only one 1 in all rows and columns) until Equation 3.1 is satisfied. Therefore, this algorithm is slightly faster for asymmetric systems, as seen in the Figure 3.10 which shows that the time to map random structures is, on average, less than the time of the more ordered structures (tree, snake, plane, centipede). The general algorithm is presented in Algorithm 2. Note that m is the size of the block redundancy in P^*; the snake example above has two blocks of two.

3.10 Complexity of the Spectral Decomposition Method

The complexity of the configuration recognition algorithm is characterized by the label mapping routine (finding the permutation matrix, Algorithm 2), since it is more computationally intensive than the configuration matching routine of finding corresponding graph spectra. The complexity of finding the graph spectra (eigenvalues)

Algorithm 2 Configuration matching using spectral decomposition.

```
for $i = 1$ to $library\ max$ do
  if $\mathrm{Det}(A_{given} - \lambda I) = \mathrm{Det}(A_i - \lambda I)$, then
    Compute $Q_{given}$ and sort (in increasing order of corresponding eigenvalue) to
    match format of $Q_i$.
    for $j = 1$ to $n$ do
      Compare the elements of columns $j$ in $Q_{given}$ with $Q_i$.
      Record $w$ as the column that produces a $P$ with the minimum number of
      redundant ones.
      if Any $P$ satisfies $A_i = PA_{given}P^{-1}$, then
        RETURN $P$.
  else
    Construct $P^*$ using column $w$.
    for $j = 1$ to $m$ do
      if Any $P^*(j)$ satisfies $A_i = P^*(j)A_{given}P^{*-1}(j)$, then
        RETURN $P = P^*(j)$.
```

of an nxn matrix using the QR-decomposition algorithm, as used for this work, is $O(n^3)$ [9].

Once a characteristic polynomial match is found, the complexity of determining P or P^* (the first if-statement in Algorithm 2) is $O(n^2)$ since the most computationally expensive loop in this step is the double loop of matching values in eigenvectors corresponding to the same eigenvector. For completely random and asymmetric structures (e.g., an arm-like robot, a head-to-tail snake, any linear or tree-like structure that is intended for forward and turning motion only), this determines P, and is consequently the expected running time to find the permutation mapping between isomorphic configurations.

For structures with at least one line of symmetry, the main if-block determines a P^* that has a P embedded within it. Such structures, such as a dog with a head and tail, a head-to-head-tail-to-tail snake (Figure 3.7), a bipedal walking robot, etc., delve into the primary else-statement of Algorithm 2 to find the correct P amongst the $m!$ choices in P^*. For the one-line-of-symmetry structures, $m = 2$, and it is evident that the complexity to find P in these cases is $O(2n)$. In general, the complexity to find P is $O(m!n)$. In most cases, m is small or at most a moderate fraction of n. In the extreme case of each module having complete symmetry with respect to another module (imagine a torus composed of CKBot modules, with each module having exactly four neighbors), $m = n$ and algorithm reduces to the naïve case of trying all possible labels for each module. But this case is pathological; m is usually a fraction of n. For example, a centipede structure with 4 identical 4-module segments, $m = 5$ (left/right symmetry plus 4 segment interchangeable symmetries) and $n = 20$.

It is certainly possible to divide the computation of reducing P^* to P amongst parallel processors. We are currently exploring a hierarchical architecture where one central processor supervises many others.

3.11 Results for Spectral Decomposition Method

Figure 3.8 shows the time to find a match using comparisons between graph spectra in a library of 200 random configurations (for each data point), for up to 1000 modules. The Matlab function $eigs(A)$ was used to find the largest eigenvalues of the sparse matrices. In comparison with Figure 3.10, we see that the time to find the module mapping is more time-consuming. This figure compares the times to find the permutation matrix between two isomorphic configurations for up to 50 modules. Snake, centipede, plane, tree and random structures were tested. The random configuration times also include a negligible matching time to find the correct structure in a library of other random structures. Even so, the cumulative time to find the mapping is slightly less than the other structures. This can be attributed to the fact that there are less symmetries (on average) in random structures and this reduces the number redundant permutation elements that the algorithm must choose from in the method described above. For comparison, the brute-force $n!$ time is included for up to 12 modules. The data point itself is off the scale of the graph. Some limiting considerations in this approach include numerical stability

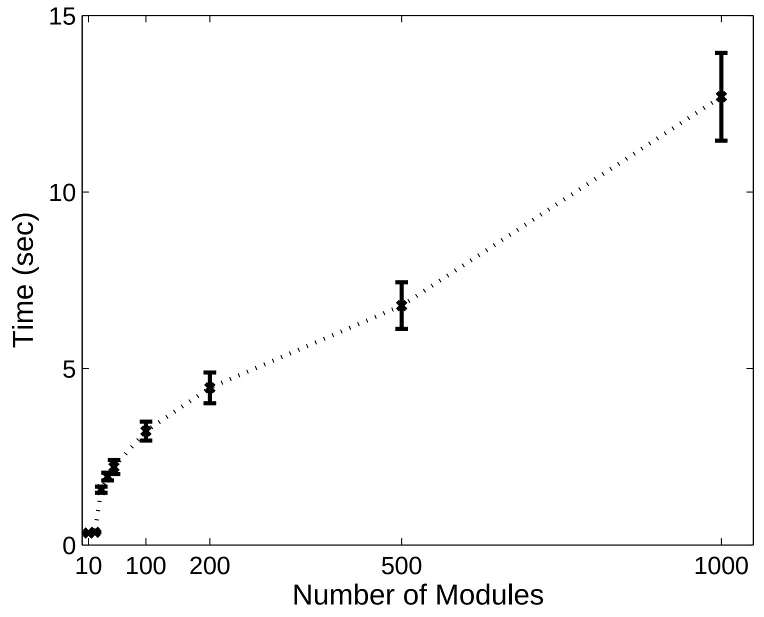


Figure 3.8: Search times for the spectral based algorithm vs. number of modules in a random robot configuration.

and structural symmetry of configurations. In particular, the number of elements of each eigenvector equals the number of modules. For large matrices, these normalized eigenvectors are composed of small numbers that have accumulated rounding errors, attributed to the LU-decomposition approach (Matlab's LAPACK matrix algebra package). There are various methods that one can implement to deal with this issue (i.e., large normalizing factors, matrix balancing) but for numbers larger than 10^3, the problem becomes difficult to handle. Additionally, the time to compute eigenvectors becomes prohibitively large at that scale.

3.12 Conclusions for Configuration Recognition

The spectral decomposition method for self-discovering CKBot configurations we have presented here is general and applicable to a wide variety of systems, as we

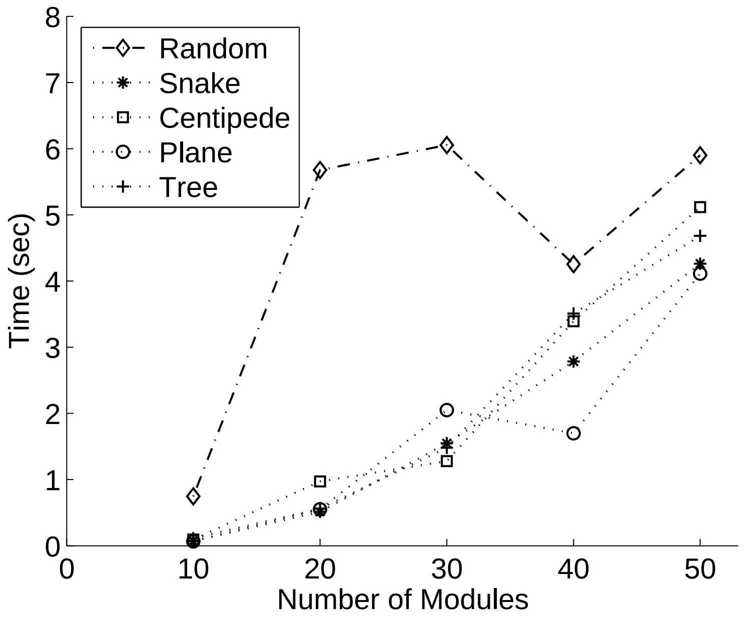


Figure 3.9: Standard deviation of search times for spectral based algorithm vs. number of modules in robot configuration.

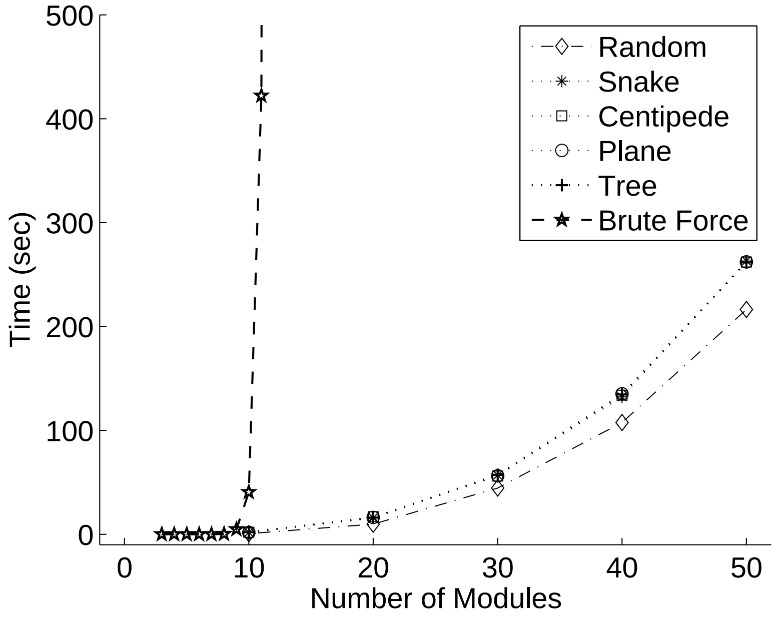


Figure 3.10: Mapping times for specific configurations vs. number of modules in configuration.

mentioned in Section 3.2. If a modular robotic system has a global bus (CAN or SPI) with labeled IR ports, then the approach we have described is directly applicable. For example, CONRO modules [4] have both a global CAN and four distinct inter-module connections that can be confirmed with IR. The configuration recognition method as described can be applied to perform calculations and structural matches with spectral graph comparisons.

If a modular system has only local IR communication (e.g., Crystalline [36]), an adapted version of our method can be applied. For configuration recognition with neighbor-to-neighbor only communication, each module in the structure must have the same configuration information. In particular, modules in this distributed system would asynchronously communicate with neighbors to acquire connectivity information using a token passing method. Then, after all modules have determined where neighbors are, modules must then propagate this information throughout the structure until all modules know the complete configuration. Finally, each module independently would then compute the spectral analysis as described in this chapter. Afterward, each module would know where its particular location is within the overall connected structure and select the appropriate control associated with that position.

Chapter 4

Distributed Communication Fault Tolerance

An advantageous feature of modular robots is the applicability to system fault tolerance. With redundant units and parallel processing facilities, modular robots have the capability or potential to account for certain types of failures within its system. One type of fault tolerance is self-repair, as we will present in the following chapter and also demonstrated on other modular robotic systems as we will describe in Section 5.1. In these works, the fault tolerance is based on assessing structural failures within the system and designing the system to repair itself with the original parts or with replacements from the environment.

In this chapter, we present work on communication fault tolerance for the CKBot [32]. Unlike the structural fault tolerance, this work is based on accounting for sensor or processor failures (including modules with perhaps incorrect or outdated software) where communication lines may be broken or bad data is passed through the system. This approach exploits the parallel processors of the CKBot with a distributed voting scheme, to override bad data or modules with faulty sensors or somehow faulty processors.

In particular, this chapter 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. This method, known as *triple modular redundancy*, was proposed by Von Neumann in the 1950's and has since been treated extensively to improve the reliability of digital systems [23].

The primary contribution of this work is a more robust, fault-tolerant modular robot. As modular robotic systems become composed of increasingly many modules, error handling becomes a crucial task. This chapter proposes a new way to handle a few faulty modules with hardware or software errors within a larger group of properly functioning modules. This method is demonstrated on a couple of CKBot configurations and shown to be useful for accounting for the occasionally erratic IR communication system.

We begin this chapter with an overview of related work in this area and statistical motivation for our work, followed by a description of the experimental system and software implementations, and conclude with experimental results, actual modes of failure in the robot and algorithm design considerations for communication fault tolerance.

4.1 Related Work

In the field of modular robotics, there has been a substantial amount of research on distributed algorithms to demonstrate self-repair, self-assembly, and self-replication. Perusing through the literature, one finds that the approaches are quite varied among the different research groups. In this section, we set the stage for our work and present a few of the approaches related to distributed control and fault tolerance of modular robotic systems.

In 2002, researchers from the Crystalline and M-TRAN projects collaborated on work on a distributed algorithm for a generic type of cubic modular robot [2]. This paper presents a method for modular robot control using a set of cellular automata neighbor rules. Simulations of this algorithm show how the system handles locomotion on flat surfaces and over simple obstacles. Distributed locomotion control is a similar feature between this work and ours; a distinction we make is the demonstration of communication fault tolerance for a subset of modules that, if not taken into account, would render the system useless. A modified version of our fault tolerance redundancy check could be incorporated into the cellular automata routines. For instance, if module diagnostics (e.g., computational and communication verification checks) were shared between modules, in addition to locomotion rules, a configuration of modules could detect and resolve critical errors.

A different type of distributed locomotion method based on local message propagation was proposed and demonstrated on the CONRO modular robot [45]. Caterpillar, sidewinder, and rolling track gaits are achieved from neighbor-to-neighbor synchronization commands. These messages spread and trigger individual module motions based on locations within the overall structures. This is a similar feature to the work we will present here where all modules in a configuration contain global gait information but select particular motions based on positions in the structure. Other common characteristics are gait synchronization and accountability of dropped messages. For this CONRO robot experiment, an occasional missed message is not critical as the next message will re-synchronize the out of phase modules. Key distinctions from this work are that modules in our experiment can handle chronically missed messages, respond to external signals, and choose from a variety of gaits based majority votes. Also inter-module message passing in our case is global-BUS whereas for the CONRO work it is entirely local. A more recent publication from the CONRO group [38] shows developments in the robot algorithm's tolerance for self-reconfiguration and selection of gaits based on local neighbor feedback without requiring unique module labels. An improvement our fault tolerant approach could offer to this work is a method for inter-module data verification. In particular, if modules in a CONRO configuration could check important features like gait synchronicity and configuration states with their neighbors, the overall system would be more robust to timing problems and intermittently dropped or corrupted messages.

One feature of the fault tolerance we present here involves functional modules overriding control of faulty ones. That is, if a module receives an incorrect command that the majority of modules do not, the minority command is replaced with the majority decision and hence individual module commands are corrected by the majority. With a similar philosophy, researchers at Johns Hopkins University presented a paper on cooperative diagnosis of faulty modules as a first step toward repair of these broken units [20]. In this work, an off-board computer equipped with an overhead camera observes the trajectories of two Lego Mindstorms robots programmed to roll along a surface in circular motions. The computer uses the camera observations to correct the motor control of the robots to adjust the trajectories toward pre-defined states. The off-board computer is considered a member of the modular robotic system, so in this way, one module is diagnosing and repairing the control of another module. Our experiments are different in that overriding control between modules is reversible (not one way directed from PC to Lego robot) and is based on coordinated group locomotion in the presence of communication faults instead of a convergence toward optimized trajectories. An application of our work to this group's approach would be the integration of a complete inter-module check of states. This would allow a greater level of system robustness and reduce the central dependence on the sole observations of the off-board computer.

Yu and Nagpal at Harvard recently presented work on an environmentally adaptive modular robot [57]. In the experiments described, each modular unit in a truss-like structure uses an accelerometer to adjust appendage length to keep the horizontal truss levels parallel to the locally flat surface of the earth. Since each module adjusts itself according to its accelerometer observations, the robotic system is completely distributed. This work is closely related to ours in that each module observes a global signal (theirs: gravity, ours: IR signal) and the overall system behaves in a deterministic coordinated behavior without a centralized controller. The key difference in our work is the sharing of the locally observed signals to decide upon a majority action and override control on the minority modules with dissenting observations. In this way, our system tolerates a minority of sensory failures; by contrast, if one module's sensory input is faulty in the aforementioned approach, then the system as a whole behaves in an unpredictable manner. A key improvement to this environmentally adaptive robot would be an accelerometer redundancy check between modules, similar to the IR redundancy check we will describe in this chapter. That is, modules might share accelerometer readings with connected units so that multiple observations can be taken into account for adjusting the robot tilt. The approach would be particularly relevant for situations where individual accelerometer readings are significantly different from connected neighbors in a way that may not be kinematically feasible; one resolution for this scenario might be to let the robot adjust its own control based on some interpolation of the neighbor observations.

4.2 Statistical Advantage of Majority Based Decisions

It is intuitively understandable that when two or more processors in a modular robot agree upon a globally observable signal this observation yields a greater confidence than if just one processor observes the signal. It is also clear that a unanimous decision amongst multiple modules would give the greatest confidence in an observation, but such a strict requirement becomes increasingly unlikely to obtain as the number of modules increases, especially considering the non-negligible failure rates of sensors and processors. In this section we present a simple model that quantifies these intuitions.

Module 1	Module 2	Module 3	Overall Probability
0 : 1/4	0 : 1/4	0 : 1/4	1/64
0 : 1/4	0 : 1/4	1 : 3/4	3/64
0 : 1/4	1 : 3/4	0 : 1/4	3/64
0 : 1/4	1 : 3/4	1 : 3/4	9/64
1 : 3/4	0 : 1/4	0 : 1/4	3/64
1 : 3/4	0 : 1/4	1 : 3/4	9/64
1 : 3/4	1 : 3/4	0 : 1/4	9/64
1 : 3/4	1 : 3/4	1 : 3/4	27/64

Table 4.1: A list of all possible states and the corresponding probabilities for three modules observing a message (*observation* : *probability* for each module). 1s denote successful observations of the transmitted IR message and 0s denote incorrect or missed observations. Module reliability is assumed to be 3/4.

To introduce the concept, consider any arbitrary configuration of three CKBot modules. Also consider that each module in the system can observe external IR signals on its receiver ports and that each module can communicate to one another via CAN, as described in Chapter 2. Let us assume that each module has a reliability ratio of 3/4, that is, on average, modules report a correct IR observation three-fourths of the time (this is a conservative error ratio). Lastly, consider that the modules receive one byte at a time and share each observation with each other via CAN. Table 4.1 shows the possible outcomes and corresponding probabilities for the three modules for each message received. A 1 denotes a successful observation of the transmitted IR message and a 0 denotes an incorrect or missed observation.

Since we are only concerned with the number of correctly received messages (not the order), Table 4.1 shows that the chance that three modules with a 3/4 reliability all incorrectly receive the wrong byte (or miss the message entirely) is 1/64. Similarly, the chance that the same modules receive one out of three messages correctly is $3/64 + 3/64 + 3/64 = 9/64$. Two out of three correct messages is more probable with a $9/64 + 9/64 + 9/64 = 27/64$ chance. The chance that all three modules receive the correct byte is also 27/64.

As expected, these probabilities tell us that when modules are fairly reliable, it is more likely that the correct message can be determined from shared observations. In particular, if only one processor's vote were considered (centralized system), there is a 75% chance of correct observation, whereas if a *majority* decision is considered, there is a $27/64 + 27/64 = 54/64 \Rightarrow 84.4\%$ chance of *at least* two out of three modules making the correct observation. In addition, the likelihood of reaching the 2/3 simple majority decision is twice as great as the 42.2% strict requirement of unanimously correct observations.

The binomial distribution [10] encapsulates this probability rule for the general case:

$$P(k; n, p) = \frac{n!}{k!(n-k)!}\, p^k (1-p)^{n-k} \tag{4.1}$$

where n is the total number of modules, k is the number of modules in observational agreement, and p is the average reliability of the modules. However, since we are interested in the probability of success greater than some majority (i.e., the cumulative distribution), we get

$$P(n, p) = \sum_{k=i}^{n} \frac{n!}{k!(n-k)!}\, p^k (1-p)^{n-k} \tag{4.2}$$

for all integers $i > nm$, where $0 \leq m \leq 1$ is the majority ratio.

For the 7 module snake and 16 module quadruped configurations we describe later in this section, we get the following majority consensus probabilities (taking $m = 1/2$):

$$P(7, 3/4) = \sum_{k=4}^{7} \frac{7!}{k!(7-k)!}\, (3/4)^k (1-3/4)^{7-k} = 0.929 \tag{4.3}$$

$$P(16, 3/4) = \sum_{k=9}^{16} \frac{16!}{k!(16-k)!}\, (3/4)^k (1-3/4)^{16-k} = 0.973, \tag{4.4}$$

respectively.

4.3 IR Fault Tolerance Experimental System

As with the other experiments in this dissertation, the CKBot modular system as described in Chapter 2 is used for demonstration of IR fault tolerance. In particular, we use seven module caterpillar and sixteen module quadruped configurations constructed from CKBot units. Each module relies on its own processor for position control, inter-module CAN communication, and IR data sensing.

Key features for this experimental system include:

- IR receivers on the module faces receive externally transmitted data,
- External boards with arrays of IR transmitter LEDs broadcast serial binary signals,
- Modules share IR signals and vote on the majority observations,
- Algorithm is distributed and each module has identical software.

A fundamental hardware feature for this work, the IR LEDs have also been used for local (neighbor-to-neighbor) communication, ground contact detection [41], configuration recognition (Section 3.4), and docking detection (Section 5.2). Figures 4.1 and 4.2 show the two configurations illuminated by IR broadcasting boards used for communication.

4.3.1 IR Broadcaster

In this work, configurations of CKBot modules receive 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

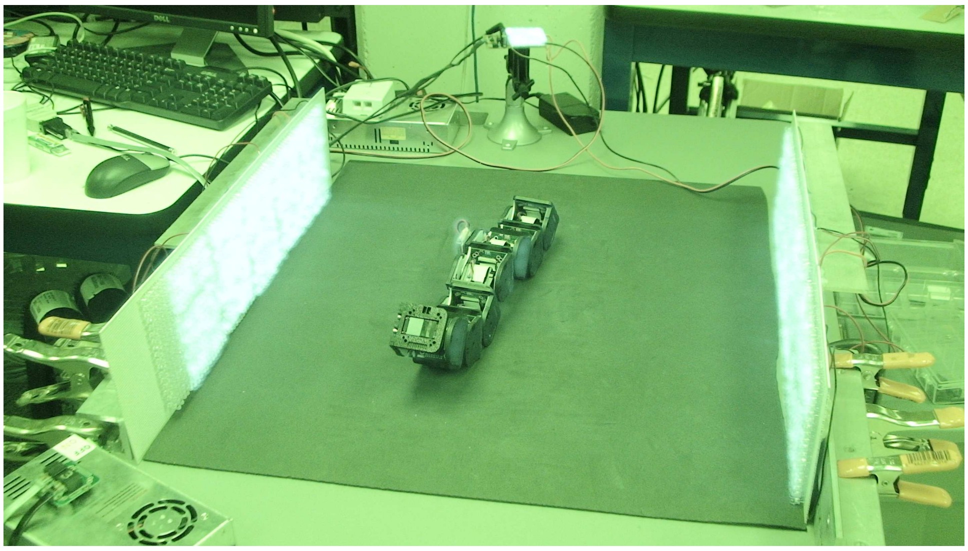


Figure 4.1: Seven module caterpillar receiving infrared (IR) signals from broadcasting boards. Digital camera in night vision mode captures the illuminated IR LEDs.

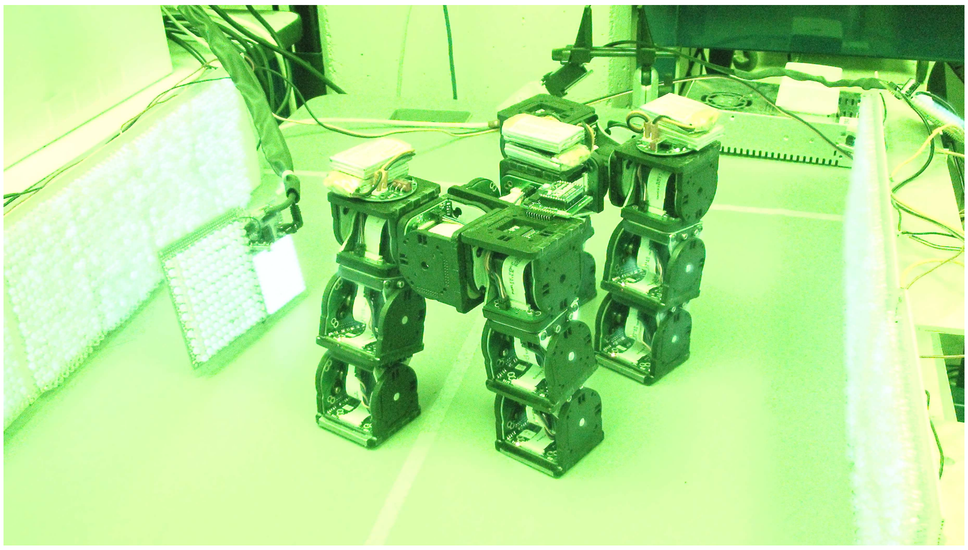


Figure 4.2: Sixteen module quadruped receiving IR signals from broadcasting boards. In this scenario, the robot must make decisions in the presence of an additional board which emits signals for commands that are different from those sent by the broadcasting boards. Digital camera in night vision mode captures the illuminated IR LEDs.

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 [33].

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 transmitted data over the given range and distances.

4.4 Distributed IR Fault Tolerance

To demonstrate distributed fault tolerance, the CKBot system in this work 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 the environment. 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. 4.1) and a sixteen-module quadruped structure (Fig. 4.2).

A majority is needed for a configuration to decide on an action; therefore, up to half of the modules in the system can be in error. Examples of errors include broken IR receivers, misinterpreted IR signals, dropped CAN messages, or any generic

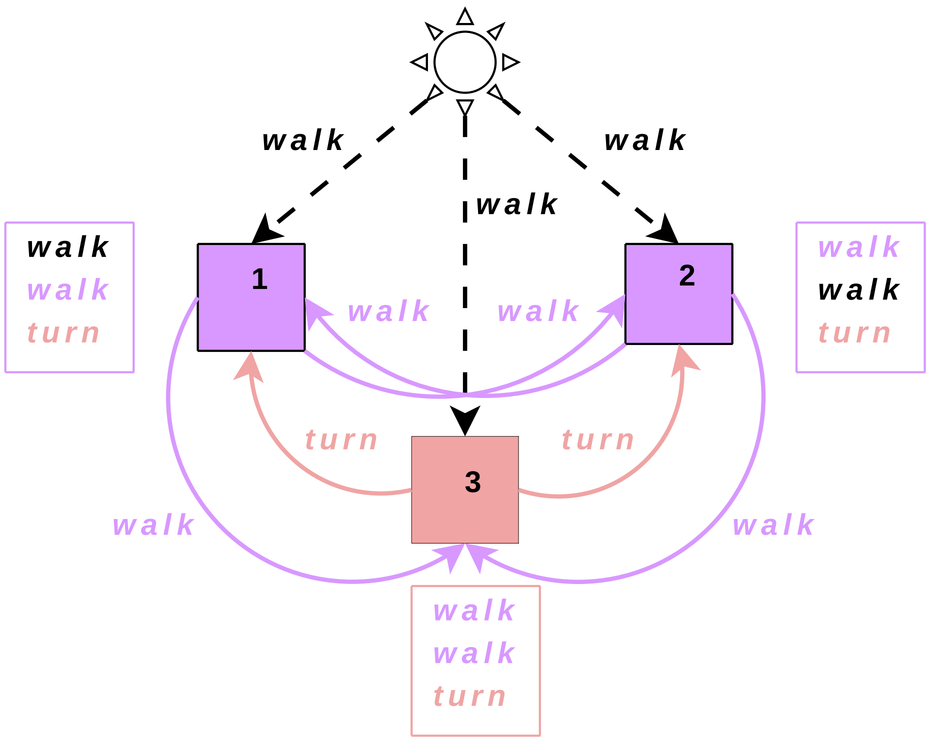


Figure 4.3: Three-module schematic of IR signal observation and subsequent sharing of data between the modules. Modules 1 and 2 correctly receive the message as "Walk," where as module 3 incorrectly interprets the message as "Turn." All modules come to the same majority decision to "Walk."

software or hardware error on modules that would affect the observation and sharing of IR data that does not disable global communications (including outdated or incorrect software on some modules). We briefly discuss a couple of modes of failure later in this section.

By accounting of the occasional error, the system as a whole is robust and tolerant to certain hardware and software failures during the course of its runtime. Fig. 4.3 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., 0b10011100 can be confused with 0b10011110). Figure 4.4 shows a schematic example of IR data being misinterpreted by a module receiver. Slowing the baud rate to widen the analog inclines/declines with respect to the overall signal packet mitigates some of these issues but does not solve them entirely (e.g., in our system we use the relatively slow serial baud rate of 2400 bps, and these errors still occasionally occur).

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. Figure 4.5 shows a photograph of an IR receiver flaking off of its PCB surface mount. These were the two most common IR communication errors.

Fig. 4.6 illustrates the fault tolerance state machine on each CKBot module's processor. To reiterate, the system as whole requires that a majority of the modules agree on the same observed value. If less than half of the modules receive inconsistent IR data, the system as a whole is not affected and the structure with this algorithm can carry out its designated task.

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.

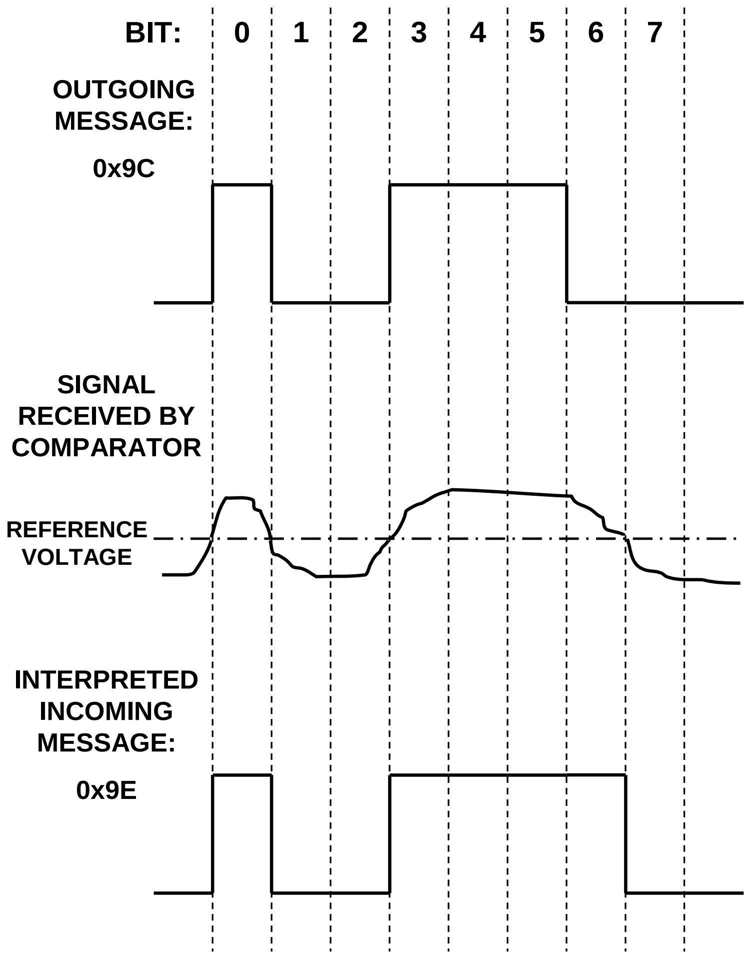


Figure 4.4: Schematic example of a module misinterpreting a serial IR byte due to an analog to digital conversion error.

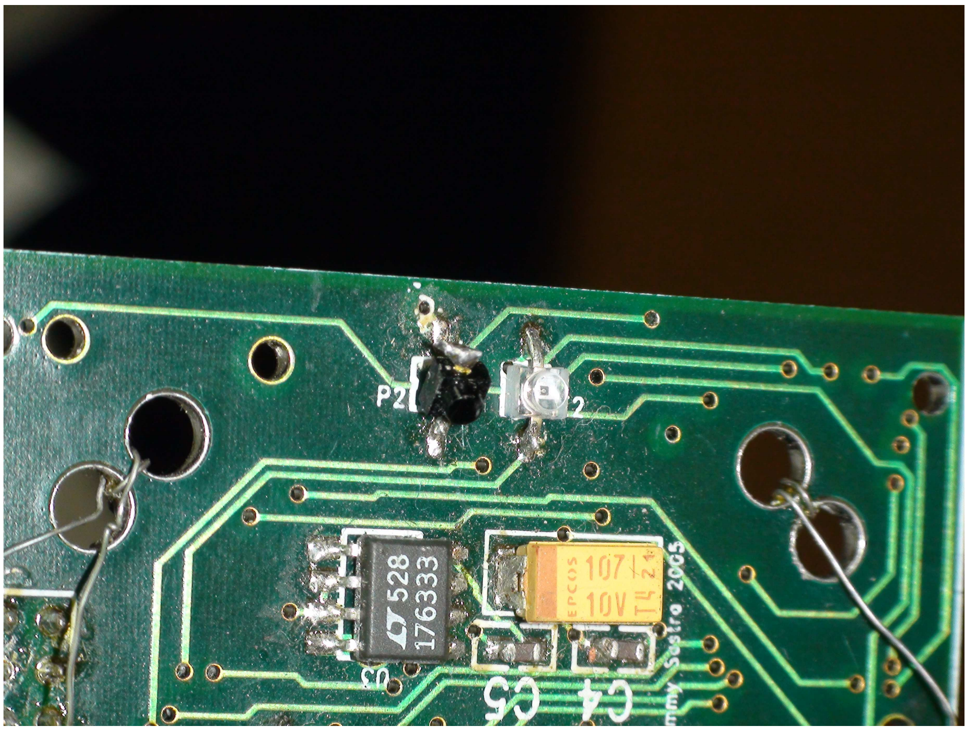


Figure 4.5: Photograph of black IR receiver flaking off of its surface-mounted position.

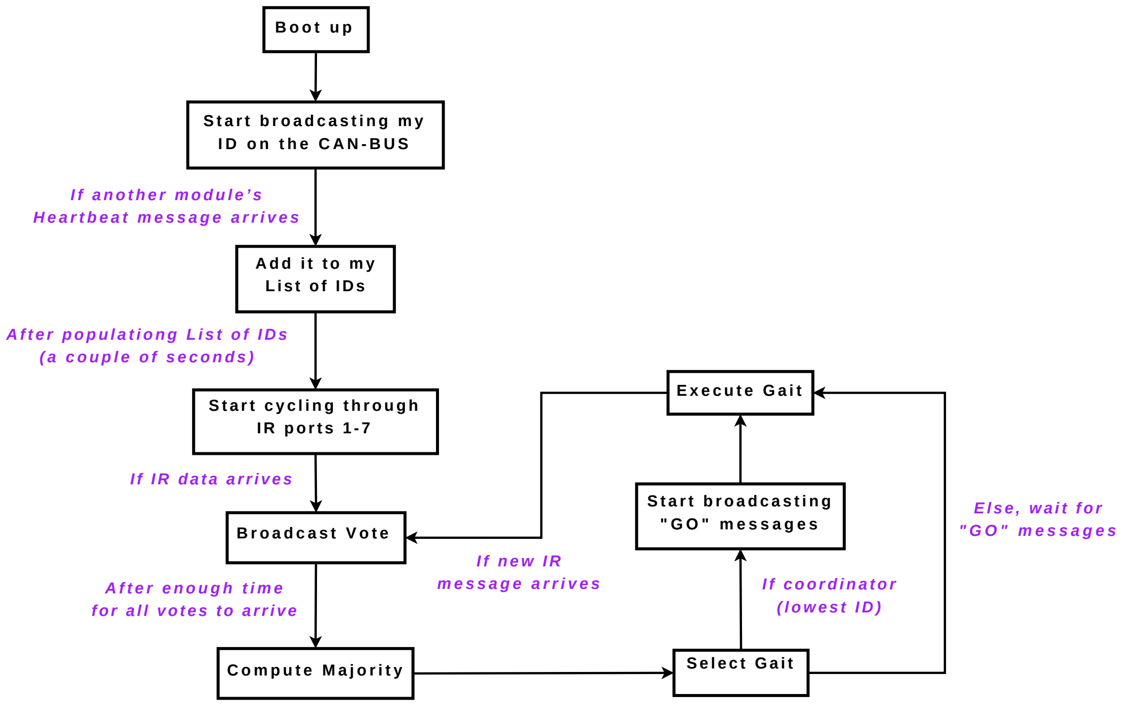

Boot up
Start broadcasting my ID on the CAN-BUS
If another module's Heartbeat message arrives
Add it to my List of IDs
After populationg List of IDs (a couple of seconds)
Start cycling through IR ports 1-7
If IR data arrives
Broadcast Vote
After enough time for all votes to arrive
Compute Majority
If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

Figure 4.6: Flow chart diagram of the processor state machine on each CKBot module.

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 ID heartbeats (as described in Chapter 2) that all modules broadcast on the CAN. Since the heartbeat frequency is 1 Hz, 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 (individual 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.

Since no assumptions are made about the source locations of broadcasted messages, the modules are designed to handle incoming IR signals on any of its four faces. In particular, each module cycles through its 7 ports to ensure that broadcasted messages from any direction are received (recall from Chapter 2 that the 7 TX and RX IR ports are multiplexed to the one pair of TX and RX USART pins on each CKBot's PIC processor). To allow sufficient time for guaranteed data reception, each CKBot cyclically selects one of its seven ports for 200 ms at a time while the broadcaster transmits 14 identical messages in series every 100 ms. 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 a gait (each module's position updates at a rate of 60 Hz) is designated. The coordinator broadcasts synchronized "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 arbitrary 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. 4.2) 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 simply an ordered mapping are useful here (i.e., actual IDs 0x12, 0x15, 0x23 mapped to virtual IDs 0x01, 0x02, 0x03, respectively). Recall that virtual node IDs are also used for ordering the rows and columns of port adjacency matrices as described in Section 3.5. Each module

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,

Figure 4.7: Walking gait control table for the 16 module quadruped. Each module contains this information and at runtime selects the appropriate column according to its local position within the overall structure.

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 reordering of module arrangements do not affect the overall motion of a given fault tolerant system, as described in Chapter 3. Such a system would automatically map the position specific gaits to relax the requirement from an ordered mapping to any isomorphic mapping for known configurations.

As a side note, since all modules for this project have identical programs, we employ a CAN-driven programming scheme which re-flashes the program memory for 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.

4.4.1 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. 4.7 shows the gait control table for the walking configuration (Fig. 4.2) 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. 4.7 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.

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.

4.4.2 Experiments

To demonstrate distributed IR fault tolerance, we implemented the algorithm described in the previous section on the caterpillar and quadruped configurations. The opposing IR broadcasting boards allow an approximately two-square-foot range in which the caterpillar and quadruped can crawl. For monitoring the status of the modules, we used a ZigBee wireless device that relays the various the inter-module CAN messages to an off-board PC.

With a hand-held joystick controller, we mapped gait commands to the IR broadcasting boards, which the modules use to decide upon actions. An external hand-held "jammer" IR broadcasting board as shown in Fig. 4.2 is introduced for the quadruped configuration to intentionally add erroneous commands to the system.

This jammer board sends commands different from those sent by the main broadcasting boards, thereby occasionally confusing a subset of modules within the configuration. With this additional device, we are able to verify and test the efficacy of the majority function routines in the modules.

An action "weighting" system was also implemented in which the *number* of modules in agreement scaled the amplitude of the corresponding actions. For instance, if only five modules in the quadruped observe the command "walk" with all others seeing nothing, the system executes the walk gait at 5/16ths the full speed and motion amplitude of the "walk" gait. As more modules confirm the same observation, the action is scaled upward accordingly. Note that for this experiment the majority threshold is not rigidly locked at 1/2; rather the number of modules in agreement is reflected in the *enthusiasm* in choice of action.

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, Fig. 4.8 shows the number of correctly received and missed IR messages versus angular position for the "limp" command. Note that for the limp command we use the byte $0xFF$ (binary $0b11111111$) and 0° corresponds to the quadruped torso parallel to the two stationary broadcasting boards. The graph shows an expected periodic pattern where 13 of the 16 modules correctly receive the command within a roughly 30° deviation from parallel or anti-parallel to the broadcasting boards. As mentioned earlier, the reason for the 3 modules missing the signals is because 3 of the 4 torso modules are not exposed to the IR signals in this configuration (refer to Fig. 4.2). The dotted line in Fig. 4.8 shows the number of missed messages for the limp command with respect to angular position. Since no messages are misinterpreted for this command, the number of missed messages is simply the difference between the total number of modules in the configuration and the subset of modules that correctly receive the messages. The couple of modules that correctly receive the messages around the perpendicular orientations (90° and

270°) are attributable to those end torso modules that have IR ports exposed to the broadcasting boards in these states.

Similarly, Fig. 4.9 shows the number of correctly received and missed IR messages versus angular position for the "stop" command. Note that for the stop command we use the byte $0xEE$ (binary $0b11101110$) and 0° corresponds to the quadruped torso parallel to the two stationary broadcasting boards. The graph shows a similar periodic pattern corresponding to the limp command (Fig. 4.8). For this stop command, however, the IR messages are sometimes misinterpreted, especially at glancing angles around 45° and at dead-on parallel and perpendicular positions as shown in Figs. 4.9 and 4.10. The reason for the incidence of incorrect observations for the $0xEE$ byte can be attributed to an increased sensitivity to misinterpret the broadcasted data as something different but closely related. For instance, $0xFF$ is not as likely to be confused with any other signal since all of the bits in this byte are high; whereas, $0xEE$ has two 0 bits whose placement may be misinterpreted by the CPU which may read $0b11101110$ as $0b11011101$ (bit shift) or $0b11101111$ (0 bandwidth too narrow), or something similar, especially at glancing angles. Figure 4.4 described earlier shows a general schematic of this effect. The inverse spikes at the precisely parallel and perpendicular orientations are similarly attributable to errors in bit detection (likely from over-saturation due to strong IR intensity at dead-on IR receiver positions). Ultimately, the sensitivity of the IR hardware is accountable for the analog-to-digital conversion error; this fault tolerant method allows for a manageable amount of hardware inaccuracy, as hoped.

Lastly, we note that the turning commands are useful for the quadruped configuration, as it allows a user the ability to control the robot to stay within orientations where the majority of modules can correctly receive the various IR messages.

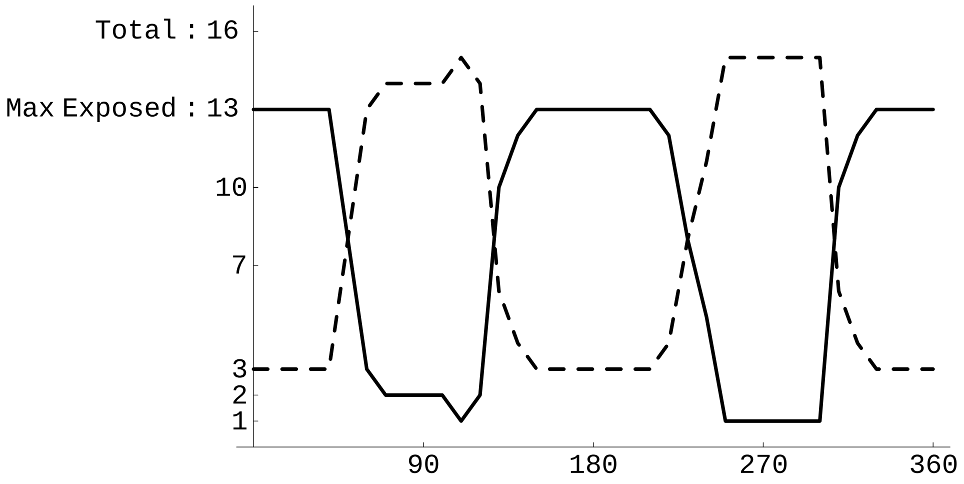


Figure 4.8: Correctly received (solid) and missed IR messages (dashed) versus angular position for the "limp" command. Note that for the limp command we use the byte $0xFF$ (binary $0b11111111$) and $0\,^\circ$ corresponds to the quadruped torso parallel to the two stationary broadcasting boards.

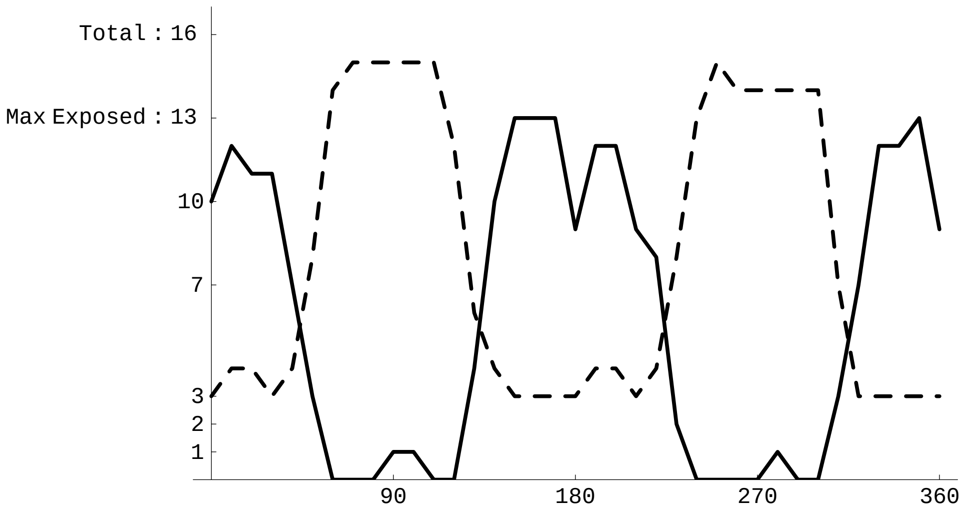


Figure 4.9: Correctly received (solid) and missed IR messages (dashed) versus angular position for the "stop" command. Note that for the stop command we use the byte $0xEE$ (binary $0b11101110$) and $0\,^\circ$ corresponds to the quadruped torso parallel to the two stationary broadcasting boards.

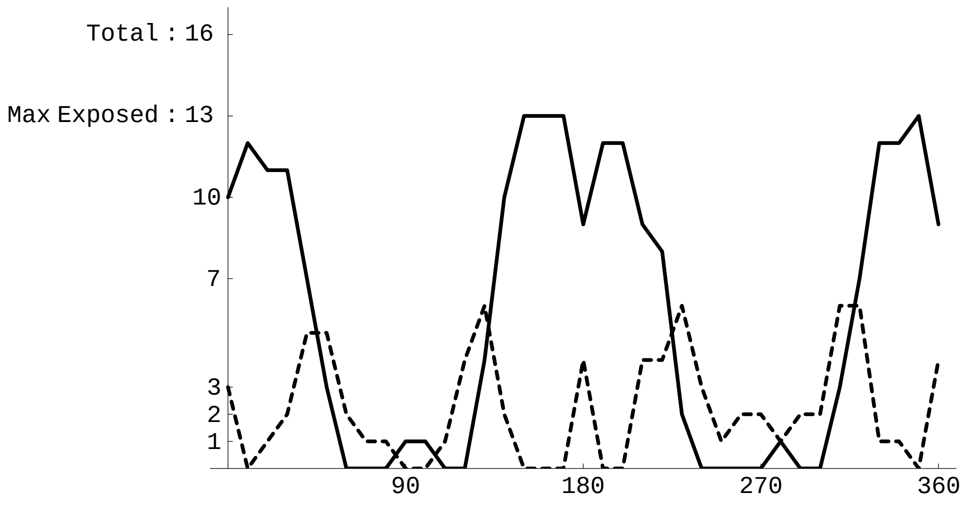


Figure 4.10: Correctly received (solid) and misinterpreted IR messages (dashed) versus angular position for the “stop” command. Note that for the stop command we use the byte $0xEE$ (binary $0b11101110$) and 0° corresponds to the quadruped torso parallel to the two stationary broadcasting boards. Also note the increased occurrence of misinterpreted signals around glancing angles (45°, 135°, etc.) and dead-on positions (0°, 90°, etc.).

4.5 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 systems where individual processors within a system all may observe the same data.

As mentioned in Section 4.1, the environmentally adaptive modular robot uses individual accelerometers in each of its modules to detect the direction of gravity, and consequently, adjust itself to provide flat surfaces on uneven terrain [57]. However, if any one of the sensors is faulty, the overall robotic structure will likely be affected in an unintended manner. If the modules in the environmentally adaptive robot are given the capability to communicate with one another, the robot could apply our fault tolerant method to override the occasional faulty accelerometer. In particular, if one module in the center of other modules were to give a gravity reading that is drastically inconsistent with its neighbors' readings (or not physically conceivable given the robot kinematics), the neighboring modules could interpolate its sensor readings to overrule the inconsistent module's control.

Similarly, camera systems as implemented on various modular robotic systems (e.g., [22], [27], [55]) are well-suited for this modular redundancy fault tolerance. Instances where modules are equipped with cameras to observe their common environment can use redundant visual information to provide more reliable localization information and more robust guided locomotion.

A benefit of this distributed algorithm is that both computation and number of messages scales with the number of modules in a configuration (N-modular redundancy). 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 redundancy 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 in the quadruped 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. One way is to use configuration topologies to distinguish modules, as no two modules can occupy the same position at the same time in a connectivity graph. However, distinguishing features must be used to disambiguate symmetries. For instance, if modules with no IDs are used to find a majority over a wireless network, they might first use neighbor connectivity to determine what kind of modules they could be; afterward, the modules would add signatures on messages, such as "Leg module," "Foot module," and so on to determine precise locations within structures.

Quantifying levels of reliability poses interesting questions. How confident should the system be in correctness of messages? What is an optimal level of confidence threshold 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 limiting 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 communicate correctly, which is superior to the requirement that one designated module that must communicate properly 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 [21]. 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 [5].

Table 4.2 summarizes the various advantages and disadvantages for the aforementioned types of majorities used for modular redundancy fault tolerance.

An interesting possibility for future work on this project would be taking the correlation of dissenting votes into account. That is, intuition tells us that a 3/5ths majority consensus with two distinct dissenting votes gives a higher level of confidence in an IR observation than if the same majority had two identical dissenting votes. Considering the Hamming distances of the dissenting votes with respect to each other and the majority observations may also give further insight into diagnosing the type and severity of errors. Accounting for such elements would help gauge the confidence in the majority decision making processes.

4.6 Conclusions for Modular Fault Tolerance

In this chapter we described 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. The binomial distribution is shown to give a measure of the statistical advantage in majority decisions for this approach and some common modes of IR communication failure are

Number of modules used for Decision	Type of Agreement	Advantages	Disadvantages
1	Centralized	Need just one correct signal	If single vote faulty, whole system fails
1/2	Simple Majority	Fairly high confidence in agreement	Ad-hoc majority
2/3	Byzantine Agreement	Tolerates intentionally incorrect inter-module messages	Bandwidth intensive; not required for most systems
ALL	Complete Consensus	Highest confidence in decision choice	Just one broken module and whole system fails

Table 4.2: Summary of the limiting and special cases for majority decision making with modular redundancy fault tolerance.

identified and discussed. We have found this model of fault tolerance to be practical and readily applicable to modular robotic systems with a global communication bus as a method to detect and correct small errors in large configurations.

Chapter 5

Self-reassembly After Explosion

Two decades after the introduction of modular robotics, some of the leading researchers in the field came together and reviewed the various research activities as well as defined some of the challenges and opportunities for progress of modular robots [54]. An overarching theme in this discussion of challenges is independence and robustness: self-repairing, self-sustaining, self-replication, and self-extension were cited as some of the goals. A small amount of progress in this work has been made as of this date and work continues toward these advancements.

In this chapter, we present some of the inner workings of a self-repairing robotic system called Self-reassembly After Explosion (SAE) with CKBot. In particular, our discussion will focus on the integrated communication hierarchy and reassembly sequence planning. A full, general presentation of the work can be found here [55] and a more in-depth discussion of the camera localization work can be found here [48].

The communication and control structure we will present is readily applicable to many systems with similar, common hardware features such as accelerometers, IR communication, smart cameras, and simple motors for motion. The primary contribution of the work here is the demonstration of a model system for organizing and controlling a complex robot that can reassemble itself after a random, destructive event.

Self-reassembly after an explosive event is a phenomenon that does not occur often in nature, as we know that statistical thermodynamics tells us that things tend toward disorder, and once disordered, it is unlikely for the disordered system to return to its original state. Biological systems resist this disorder while alive, and when recoverable, require relatively high amounts of energy to return the state of order. The more general scenario is for things to become disordered and the constituent parts become dispersed into parts of other systems (decomposition/metabolism, mass from star explosions and gravity).

If a system is able to reform after disassembly, sometimes it is desirable for it to break apart during an unexpected, destructive event. The energy dissipated from the breaking of bonds absorbs some of the energy of the destructive force. Examples of this include car bumpers that crumple upon impact, absorbing some of the shock of impact and ski boot bindings that come apart to avoid human injury from a fall.

In this work, we apply the concept of structured disassembly and self-reassembly to our CKBot system to demonstrate robustness of a modular robotic system. To demonstrate this, the CKBot is shown to recover from an interruptive, destructive event. Figure 5.1 shows an example sequence of events. In summary, a walking configuration is suddenly kicked into three separate pieces. The three clusters orient themselves and find one another with cameras. The clusters then proceed to move toward one another and reconnect at the magnetic bonding joints. Once reconnected, the structure stands up again and continues its original task of walking.

During the sequence of events, the modules all require communication and coordinated control based on various inputs from each other the environment. In this chapter, we discuss how our system incorporates the various inputs into its control loop as well as how the system communicates at all points in its runtime.

Achieving this work required integration of many technical aspects for CKBot, notably the smart camera, docking magnetic faces, controller sub-modules (also used

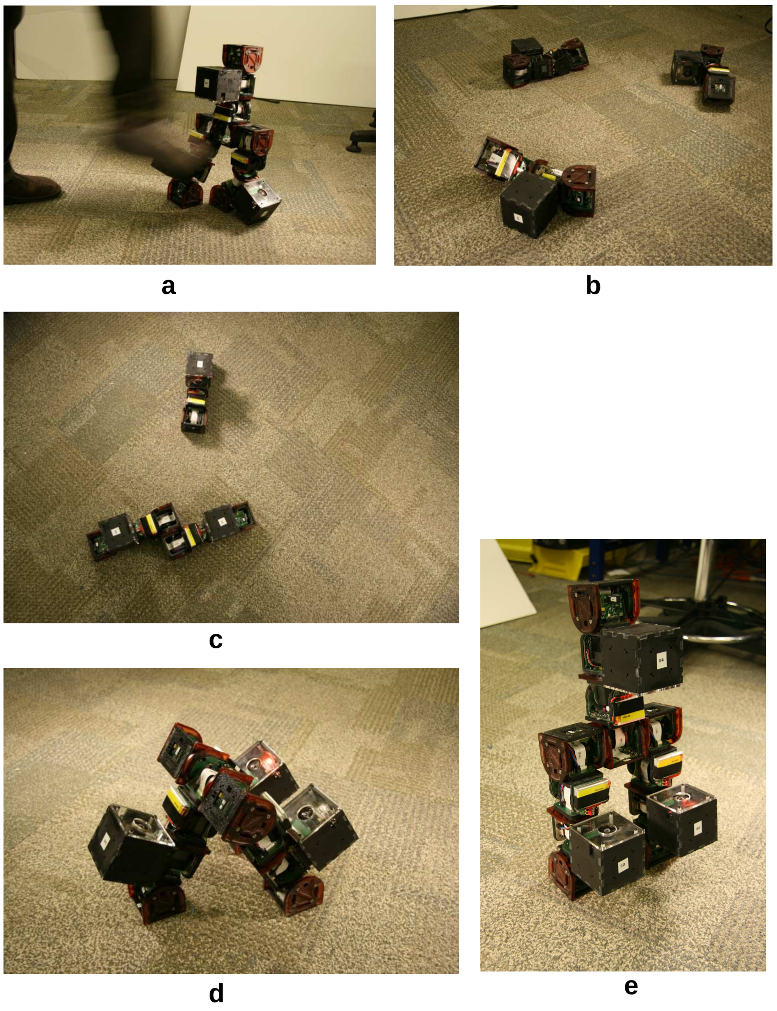


Figure 5.1: Three piece Self-reassembly After Explosion (SAE). a) kick to midsection, b) resulting three clusters of modules strewn randomly, c) clusters self-right and dock, d) system stands up, e) system resumes walking. Photos courtesy Jimmy Sastra.

in the configuration recognition work as described in Chapter 3), IR communication, as well as the CAN protocol within clusters and low-level module control. Many of these technical features will be described. In the following section, we discuss some related research.

5.1 Related Work

From the introduction of the first modular robots, a significant portion of modular robotics research has been devoted to self-repair. The analogy to organism organization on the cellular level lends modular robots to be used in studies of self-repair. In essence, various researchers have studied related aspects of self-assembly and achieved several of the elements described in self-repair.

In 1994, researchers at AIST in Japan introduced a self-assembling and self-repairing modular system [28]. Composed of identical triangular units equipped with on-board processors, *Fracta* modules connect to one another with switchable electromagnets and follow simple sets of instructions on how to assemble based on local connectivity rules. Fracta modules communicate neighbor-to-neighbor with IR signals and are able to rotate with respect to one another (on a flat surface) with magnetic forces. Key features of this work are that the algorithm is distributed and all modules control how to connect to one another in a system based on the same set of rules. Fracta also demonstrates self-repair in the context of a connected system: if a module is detected to be non-responsive (broken), it is cut off from the connected configuration and a new Fracta module is introduced to replace it. The work we will present here is distinct from this in that our system self-repairs after a destructive event that disconnects the modular system. Camera-guided locomotion and multi-step sequences for self-assembly are also distinguishing features.

One of the key features of our self-reassembly experiments is the reattachment of modular parts after disassembly and guided navigation. In our work, the docking is magnetic and the connected components confirm this connection with IR signals, as we will discuss later in this chapter. Researchers at the University of Southern California demonstrated IR guided docking with their CONRO robots in 2001 [42]. Using IR transmitters and receivers as guidance, CONRO was shown to be able to successfully dock when aligned at close range. The connection is held fixed with locking pins and disconnection is made possible with shape memory alloy (SMA) wires that release connection tabs. After docking, the connected component was shown to recognize its new configuration and carry out a different gait, more suitable for the larger structure. This is similar to our configuration recognition work presented earlier, however, without the feature of configuration isomorphism. The docking feature we use in self-reassembly is also similar in that the connected components communicate through IR; however, the our work is different in that the docking can be far range, started from more general initial conditions, and is guided using smart cameras rather than IR components.

Another type of robotic system that shares the characteristics of having modules that can move independently and also as a connected group is called Swarmbot [26]. Developed at École Polytechnique Fédérale de Lausanne in 2005, Swarmbots have the notable abilities to roll around on surfaces and connect to one another with special grippers. These grippers are also used to grasp objects and move them, as shown in a video where 35 Swarmbots pull a child across a room floor. The idea of having independent mobile modules which connect to groups of larger modules is similar to the work we will present here; however, a key difference is that our work is designed to be robust to disassembly after a high-energy event and focuses on reassembly methods as a feature of self-repair.

In 2004 Støy and Nagpal presented a method for self-repair in the context of modular robots [44]. This work shows a way for cubic modules to approximate shapes by reconfiguring to fill the volume of a desired goal structure. Simulations of modules reconfiguring to fill the volumes CAD models was shown to work following a gradient-based method to move modules into empty spaces. This type of self-repair through reconfiguration is interesting in terms of its scale (both the numbers of modules simulated and numbers of moves required were on the order of tens-of-thousands). Our work shares the idea of converging toward a goal configuration, without any gradient-based path planning but with the ability for physical connections/disconnections after perturbation of a desired configuration.

Vision-guided locomotion is central to our approach for the self-reassembly experiments. A similar method for camera-based docking in the M-TRAN modular robot was introduced by Murata et al. in 2007 [27]. This work introduces a special module with a pinhole camera that is integrated into a cluster of standard, non-camera M-TRAN modules. The method for docking using cameras incorporates LED emitters from other non-camera modules. Using observed distances between LED emitters, the camera module determines distance and orientation with simple trigonometric calculations. When close, the camera assembly of modules configures into a "docking" structure for positional error correction and to guide the approaching cluster of modules into a precise location for successful docking. This work is related to the localization feature of the work we present here; in our approach the methods for distance and angle orientation are based on pixel size and degree off-center between cameras with unique LED blinking patterns instead of geometric scale factors derived from LED distance measurements.

A couple of other research groups have presented work on self-replicating robots, inspired by concepts from self-reproducing automata pioneered by Penrose [34] and von Neumann [49] a half-century ago. The *molecube* robot from Cornell University showed self-reproduction of modular robot [60]. In this work, a pillar of four molecube modules recreates another pillar by picking up other modules from "feeding" locations and stacking them into a configuration identical to itself. In a similar approach, a group at Johns Hopkins University used Lego Mindstorms kits to demonstrate self-replication of a track-following robot with parts for duplicate assembly placed at points along winding tracks [22]. This group also describes a way to quantify the state of disorder in their self-replicating system, borrowing the concept of entropy from statistical thermodynamics. The connection mechanisms for both of these robots is attributed to permanent magnets, as is also the case in our self-reassembly demonstrations. The related concepts of self-replication and self-assembly place these works in a context similar to the work we will present here; a key feature that is different in the implementation of these concepts is their use of structured environments to introduce parts for self-replication versus our localizing and reassembling of original robot parts exploded into random positions.

5.2 Self-reassembly System

The self-reassembly robot is built using the CKBot modular system, as described in Chapter 2. In this section, we briefly describe additional features tailored for this work.

The self-reassembly robot is made up of three identical clusters of modules. Figure 5.2 shows a photograph of one of these clusters. Each cluster is, in some sense, a self-contained unit equipped with:

- Four CKBot modules (three U-Bar types and one L7 type, as described in Chapter 2) for general locomotion and sensing inter-cluster connections;
- One sub-module controller for the central processing of the cluster and the connected robot as a whole;
- One smart camera assembly for sensing other clusters' relative positions;
- One LED for broadcasting the cluster position (inside the camera assembly box);

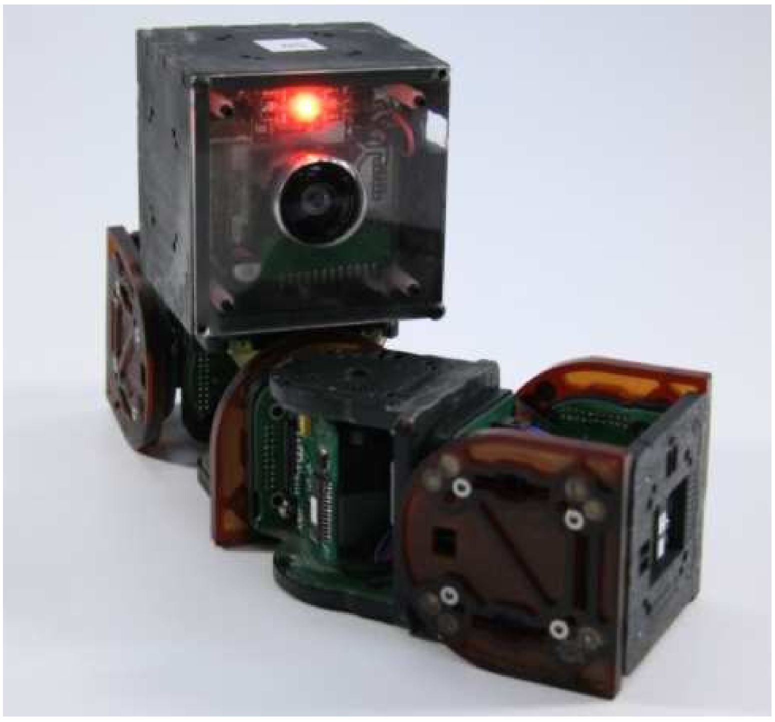


Figure 5.2: One self-reassembly cluster made up of four CKBot modules, smart-camera module, submodule controller, and magnetic face attachments. Photo courtesy Jimmy Sastra.

- One three-axis accelerometer for detecting the cluster position with respect to gravity (also inside the camera assembly box);

- Four magnet-faces (two for each of the end modules) to allow for docking and inter-cluster communication via IR;

- Two 12V lithium-polymer batteries connected in series for providing power to all components;

- One DC-to-DC converter that converts 24V to 6V power required for all the modules.

All components within each cluster are physically connected with screws and electrically connected with 20-pin headers. Therefore, all components within each cluster communicate via the CAN-bus. The structured disassembly joints are the magnetic face plates that connect clusters together. These bonds are strong enough for the robot's tasks of bi-pedal walking and reassembly sequence, but are weak enough to disconnect upon impact from a swift kick. An important feature of these magnet faces is that they allow magnetically connected modules from different clusters to communicate with one another via the IR ports. This communication link allows processors between clusters to talk with one another.

5.3 SAE Communication Hierarchy

The control structure for the self-reassembly robot is centralized with various sensor input and output commands that employ the various communication media. This structure is general and adaptable to other modular robots with similar communication systems, such as those described in Section 1.1. The overall design of this communication scheme is based on interfacing of distinct global networks with IR and smart cameras to create a connected modular robotic unit. Modular robots that change their connectivity throughout runtime, such as CONRO [3] and M-TRAN [29] can apply this communication hierarchy to their reconfigurable systems.

Figure 5.3 shows a sample image from the perspective of a camera module that detects two other clusters. In this configuration, all three SAE clusters are disjoined and in the process of localizing with respect to one another to plan locomotion for reassembly. In this section, we describe the hierarchical communication system that the robot uses throughout the reassembly sequence.

The general structure for the SAE robot is simple: three physically identical clusters (as described in the previous section) that can move around independently and can connect magnetically to each other at the ends to form a bipedal robot that stands and walks. Figure 5.4 depicts the overall connected structure for the robot. Note the cluster designations:

- Torso with modules T1-T4; Camera module TC;
- Left Leg with modules L1-L4; Camera module LC;
- Right Leg with modules R1-R4; Camera module RC;
- End Modules L4 and R4 can communicate with T4 through IR ports 3 and 5 when connected, providing the inter-cluster communication link.

There are three primary means of communication for the SAE robot. Within clusters, communication is entirely through CAN, with the various components sending messages on the global bus. Between clusters, the robot communicates with the smart camera/LED system when disjoined, and with the IR TX/RX LEDs when magnetically docked.

Figure 5.5 depicts the communication between the Torso and Left leg clusters. The component labels T1-T4, TC, L1-L4, and LC correspond to those described earlier and shown in Figure 5.4. This schematic shows how the submodule controllers are at the center of all input and output commands within clusters.

In summary, the inputs to each controller include:

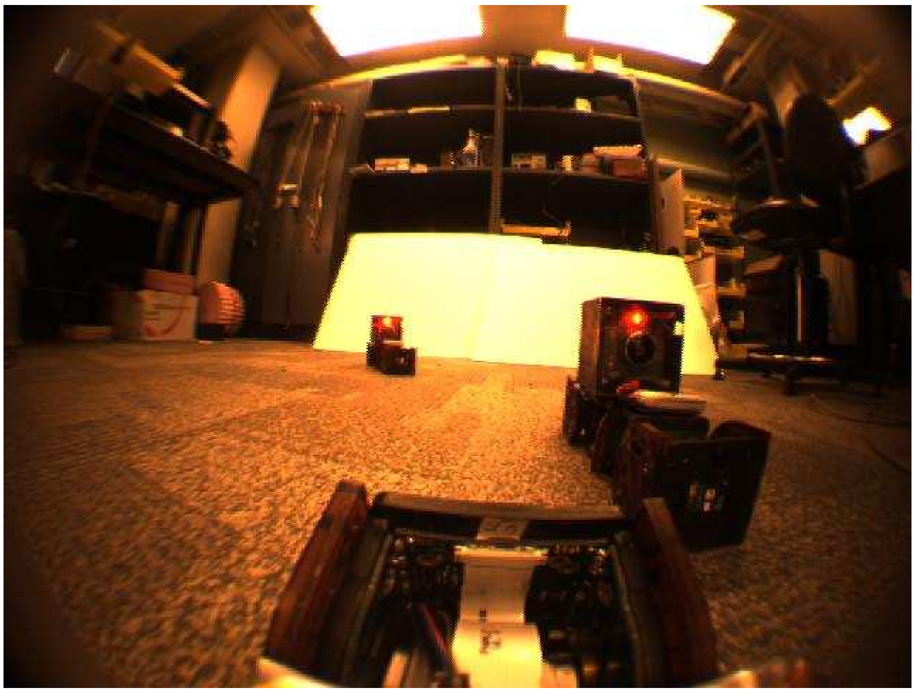


Figure 5.3: A view of two CKBot clusters from a camera module. The wide angle fisheye lens covers almost 120 degrees. Photo courtesy Babak Shirmohammadi.

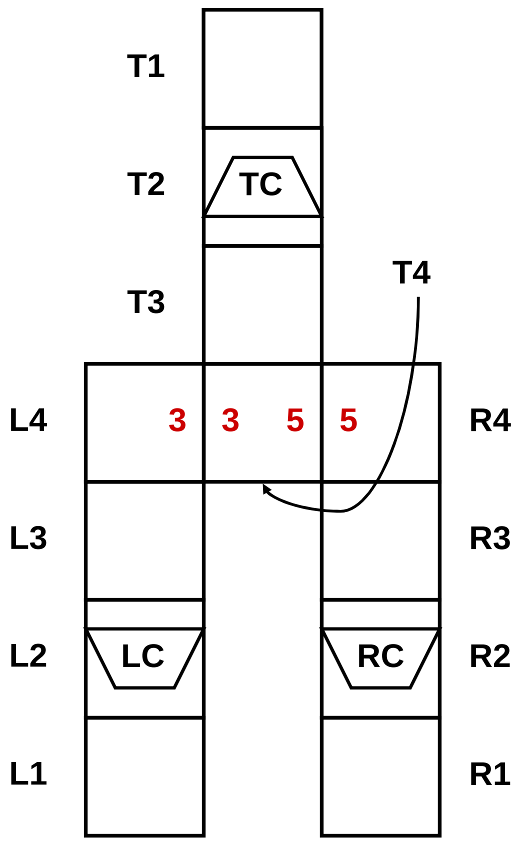


Figure 5.4: Diagram of the connected SAE Robot. Torso, Left and Right cluster Modules are labeled 1 – 4; Camera modules are labeled TC, LC, and RC. Modules L4 and R4 can communicate with T4 through IR ports 3 and 5.

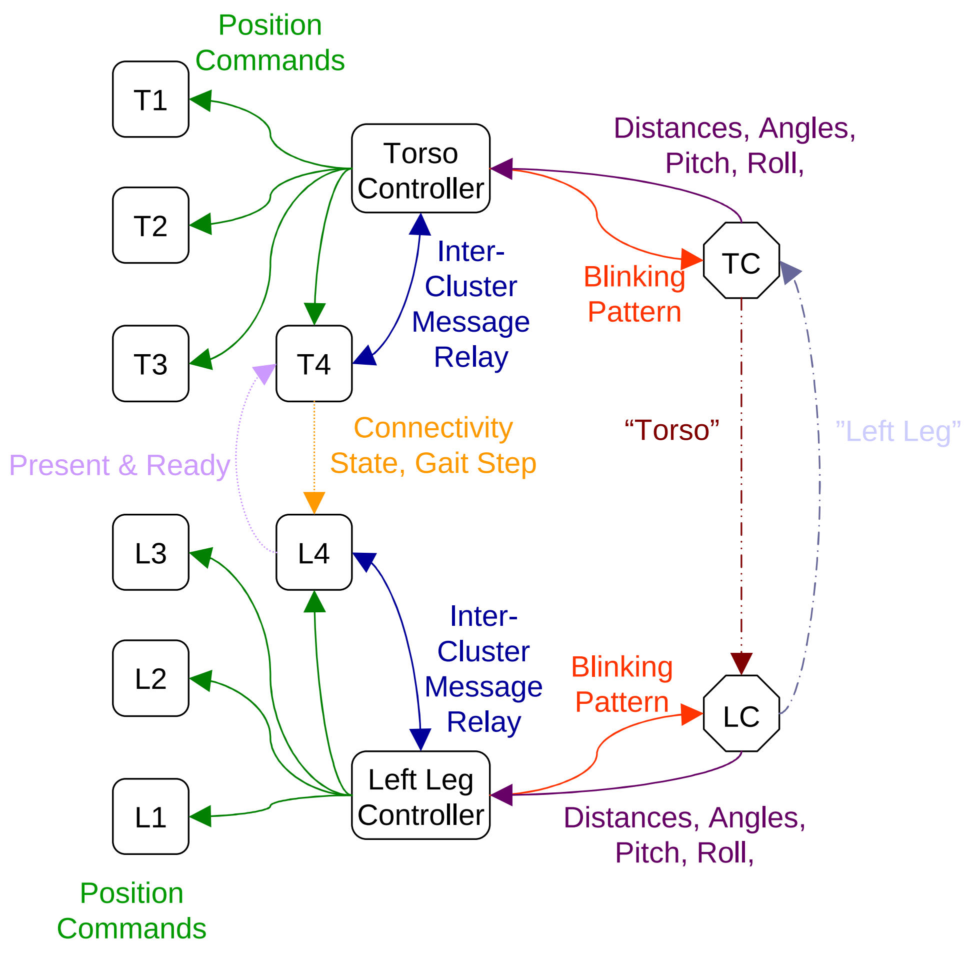


Figure 5.5: Self-reassembly communication structure schematic between the Torso and Left leg clusters. Labels T1-T4 refer to the modules in the Torso cluster, and TC refers to the Torso cluster camera. Designations for modules in the Left leg cluster are analogous (L1-L4, LC). All communication within clusters is via CAN; communication between clusters occurs with Camera (after Camera LED sighting) and IR (after physically docking).

- From camera: distances and angles to other clusters;
- From accelerometer in camera module: pitch and roll of cluster with respect to gravity (for self-righting and standing up);
- From end modules (T4, L4): inter-cluster messages transmitted between end-modules' IR ports (for confirming connections and coordinating joined cluster locomotion).

Outputs from the controller include:

- To LED on camera: identification blinking pattern. Each camera on each cluster has a unique blinking pattern used for searching and guided locomotion towards docking for clusters. Blinking pattern also changes as reassembly sequence progresses. For instance, the Torso camera will blink a new pattern after it has confirmed successful docking with one of the legs, indicating to the other leg that it is partially assembled and ready for the next docking.
- To end module: requests for connection confirmation when close to docking state; after successful docking, Torso controller sends inter-cluster relay message to other controller in leg cluster to coordinate locomotion for sequential docking, standing, and walking.
- To all modules: position commands for local cluster gait control.

Note the hierarchy of controller submodules. The Torso controller orchestrates the overall control of the system with gait synchronization and update commands to the Left and Right leg controllers. Tables 5.1, 5.2, and 5.3 show the sequence of messages during inter-cluster docking and subsequent execution of the bipedal standing gait. Note that the end modules in both the Torso and Leg clusters are given the corresponding Virtual IDs $0x04$ corresponding to Figure 5.4. This ID mapping allows the clusters to know *where* they are connected to each other regardless of specific Node ID when communicating through IR. Also, the end modules communicate which port they observe a connection to tell the controllers *how* (what specific orientation) they are connected to each other. For example, when T4 receives an IR signal $0x04$ on it RX Port $0x03$, it relays this information to the controller as $0x354$: $0x04$ $0x03$ meaning "CAN ID 54, I have received the data $0x04$ (an end module connection) through my Port 3." The CAN message prefix $0x3yy$ (where yy is the specific module ID) is simply an identifier used by the modules to distinguish the messages as inter-cluster relay messages (as opposed to position control or feedback messages).

Additionally, some specific commands are used by the controllers to end modules. The CAN message $0x3yy$: $0x08$ (where yy is the end module's ID) is a command that tells the end module to blink its Virtual ID (e.g., $0x04$) on all of it seven IR TX ports to initiate and confirm an inter-cluster connection. Some other commands not included in Tables 5.1 and 5.2 (omitted for clarity and brevity) are $0x09$ (set your Virtual ID to 1), $0x0A$ (set your Virtual ID to 4), $0xzz$ (Where zz is a hexadecimal digit from $0x0B$ to $0xFF$ that corresponds to specific gait selection codes (i.e., "Stand Up"$= 0x15$, "Turn Right 10°"$=0x22$, etc.)).

Tables 5.1, 5.2, and 5.3 contain only a segment of the complete self-reassembly cycle. Similar control and feedback loops between the controllers and peripheral devices (camera processor, camera LED, accelerometer) are integrated into the communication structure.

Distance measurements are interpreted from number of pixels detected to be blinking from the viewpoint of each camera. Figure 5.6 shows a sample mapping of distance to number of pixels, used to derive a function and calibrate the camera's distance measurement. A similar method is used to calibrate cameras to angle detection offset from the center axis.

Note that although the all the clusters are physically identical, the software within them are specialized for the roles within the self-reassembly system. This structure

Robot Component	**Torso Controller**	**Torso End Module**	**Leg End Module**	**Leg Controller**
Example CAN IDs	$0x54$	$0xA1$	$0xB2$	$0x44$
Corresponding Virtual IDs		$0x04$	$0x04$	
Communication Connections	Torso CAN	Torso CAN, IR TX/RX	Leg CAN, IR TX/RX	Leg CAN
Entering Docking Mode	CAN msg from Camera: "Leg is close enough for docking"			
Torso Searching for Connection	CAN msg to End Module: "Blink your Virtual ID on all Ports" $0x3A1 : 0x08$			
		IR TX on all ports: "I'm an End Module" IR TX: $0x04$		
Leg Receiving Search Message			IR RX Port 5: "Message from an End Module" IR RX: $0x04$	
Leg End Module CAN Message Relay			CAN msg to Controller: "Connected to an End Module through Port 3" $0x312 : 0x04\ 0x03$	
				CAN msg from End Module: "Connected to another Cluster through Port 3"

Table 5.1: First part of the self-reassembly docking confirmation sequence via IR-CAN relay messaging. This section of the cycle contains inter-cluster communication initialization.

Robot Component	**Torso Controller**	**Torso End Module**	**Leg End Module**	**Leg Controller**
Leg End Module Confirming Connection				CAN msg to End Module: "Blink your Virtual ID on all Ports" $0x3B2 : 0x08$
			IR TX Port 3: "I'm an End Module" IR TX: $0x04$	
Torso Receiving Confirmation		IR RX on Port 3: $0x04$		
		CAN msg to Controller: "Confirmation Received" $0x311$: $0x04$ $0x03$		
	CAN msg from End Module; Inter-Cluster Connection Confirmed			
Robot Assembly Starts to Stand Up	CAN msg from Accelerometer: "Pitch is about $0\,°$," Select Stand Gait			
	CAN msg to End Module: "Select Stand Gait"			
	CAN msg to all Modules in Cluster: "Start Stand Gait"			

Table 5.2: Second part of the self-reassembly docking confirmation sequence and inter-cluster gait coordination via IR-CAN relay messaging. This section of the cycle includes leg docking confirmation and self-righting stages.

Robot Component	**Torso Controller**	**Torso End Module**	**Leg End Module**	**Leg Controller**
Torso End Module Relays Stand Gait Selection		CAN msg from Controller “Select Stand Gait”		
		IR TX on all Ports: “Select Stand Gait” IR TX: $0x15$		
			IR RX on Port 3: “Select Stand Gait”	
			CAN msg to Controller: “Select Stand Gait” $0x3B2$: $0x15$ $0x03$	
Leg Controller Receives Command				CAN msg from End Module: “Select Stand Gait”
Leg Controller Commands Modules in Cluster				CAN msg to All Modules: “Start Stand Gait”

Table 5.3: Third part of the self-reassembly docking confirmation sequence and inter-cluster gait coordination via IR-CAN relay messaging. This section of the cycle includes coordination for assembled bipedal walking between the torso and left leg.

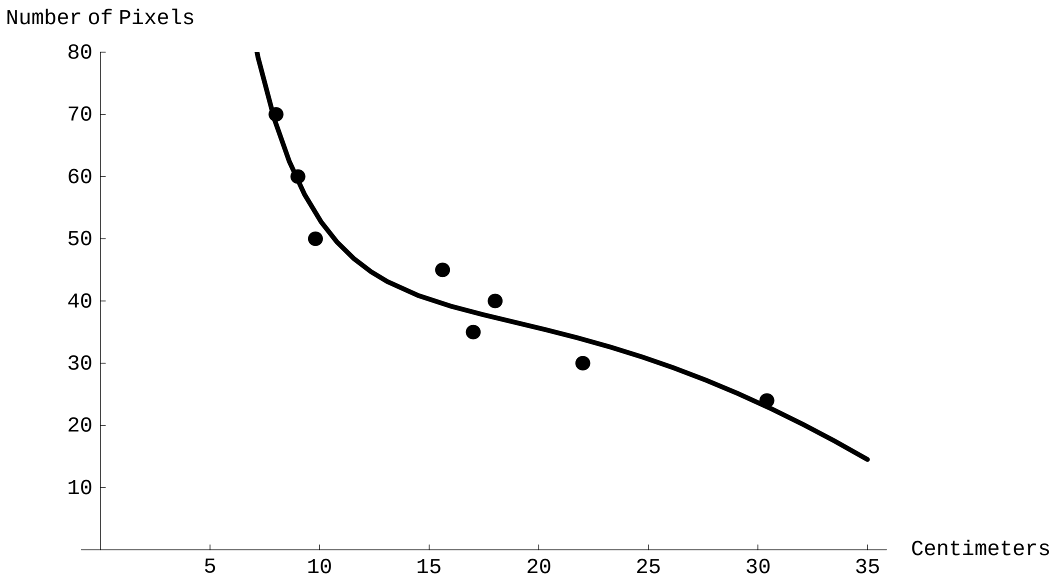


Figure 5.6: Sample data of pixels versus distance used to calibrate a camera. Dots denote pixel measurements and line denotes fitted function: $P(x) = 212\ln(x) + 6.46x + 1990/x - 568$.

is not a requirement, and we will later discuss generalizing this system to allow clusters to be fully interchangeable to be both hardware and software symmetric. In the following section, we discuss in more detail how the controllers use the inputs to control the clusters and carry out the reassembly sequence.

5.4 Reassembly Control Sequence

With the various input and output communication lines in place, as described in the previous section, it is up to the submodule controllers to organize the data and guide the modular system throughout the walking, explosion, orientation, and reassembly sequence. In particular, the controllers use a state machine to direct modules in the clusters to accomplish the various tasks required in each given situation.

The system state machine for self-reassembly is based on a synchronized guidance of modular robotic clusters toward a goal configuration and task. This principal of cooperative convergence of reassembly is general and applicable to numerous modular robotic systems that may require inter-module coordination between disjoined and connected states, such as Swarmbots [26] and Catoms [19].

Figure 5.7 shows the basic schematic for the state machine program on the torso submodule controller. This diagram summarizes the cycle of actions that the controller on the torso cluster directs during the robot runtime. In the following walk-through of the reassembly sequence, we refer to the pictures (**a-e**) in Figure 5.1 to compare with sections of the diagram in Figure 5.7. A narrated video of this sequence is also available online [25].

First, we start with the three clusters assembled, upright, and walking (picture **a**). This state corresponds the right section of the schematic where the torso controller has confirmed both leg connections and the accelerometer in the camera module tells it that the clusters are upright (pitch is between 45° and 135°). In this state, the controller sends walk commands to both the modules in its own cluster, and the

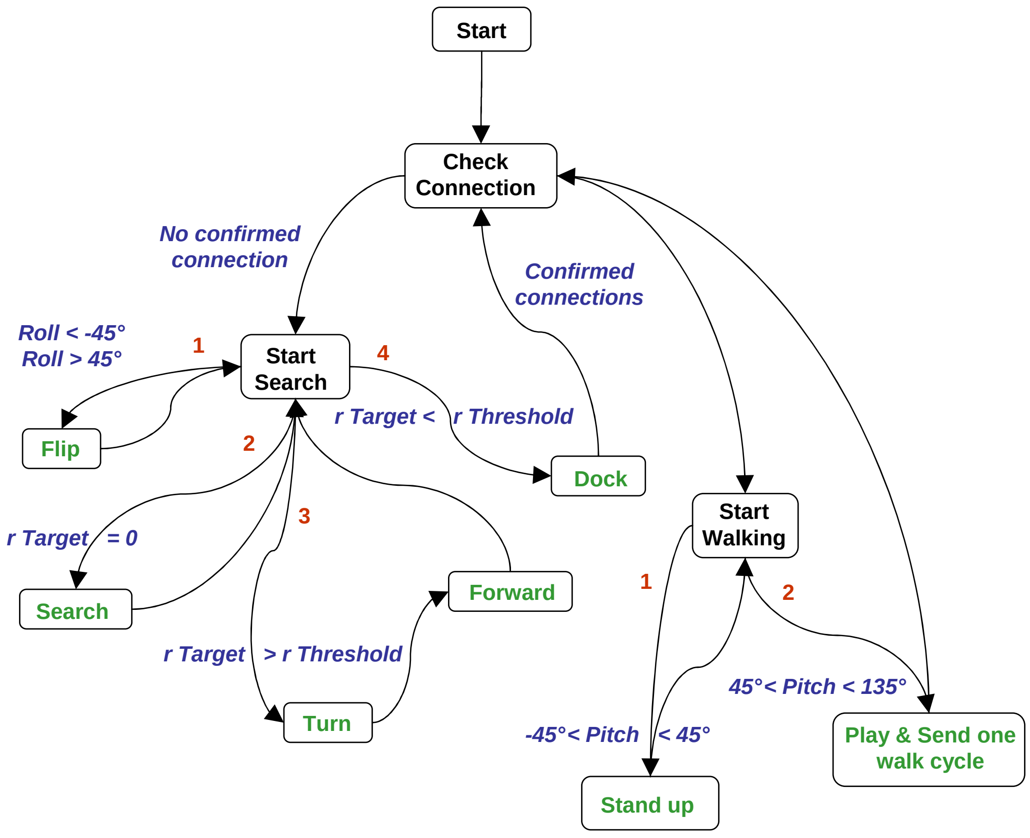


Figure 5.7: State machine schematic for the torso submodule self-reassembly controller. The sequence is a closed-loop cycle and this controller receives inputs from the camera, accelerometer, and the leg controllers via the IR connection on its end module.

controllers in the other clusters via the CAN-IR-CAN relay sequence through the end modules. The leg controllers take care of the lower-level gait commands for modules within their cluster; the torso controller's main function here is to coordinate the timing of overall motion.

After the swift kick to the robot's midsection, the system ends up in the state picture **b**, with the clusters randomly strewn apart on the ground. This brings the controller to the left section of the state machine. Here, the controller has no confirmed connections to other clusters and first checks to see its orientation with respect to the ground. If the accelerometer reports that the module roll is less than $-45\,^{\circ}$ or greater than $45\,^{\circ}$, the controller instructs the modules to perform a self-righting gait to be in a camera-up position to communicate with other clusters' cameras and to be able to move about.

Next, the controller checks with the camera module to see it it has seen any other clusters. As mentioned in the previous section, the camera reports sightings as number of pixels and angle offset of another cluster's LED blink pattern with respect the camera center axis. If the camera does not report any sightings, the controller instructs the cluster to rotate in place randomly until another cluster LED is spotted.

When another cluster has been spotted, the torso aligns itself and moves toward the leg cluster to dock. This step corresponds to picture **c** and schematic step **3** in the Start Search section. Here, the torso and leg clusters have detected each other, and both are moving to try to dock. The distance is too far to dock (r *Target* $>$ r *Threshold*) so the clusters turn and move toward each other, checking and updating their respective orientations along the way. When the cameras have reported that they are close enough to try to dock (r *Target* $<$ r *Threshold*), the controller directs a delicate docking gait to try to have the magnets of the end module faces connect (step **4** to *Dock* state in the diagram). This docking procedure is very sensitive to the camera calibration and cluster body geometry and in practice occasionally required a few tries for successful magnetic docking.

After successful docking, the end modules of the connected clusters send IR signals back and forth through the IR ports to confirm the connection (as described in the earlier section). Once this confirmation has taken place, the controller now switches its LED signal to indicate to the other leg module that it has already connected to one leg and is ready for the next docking. The connected leg cluster now follows turning gait commands from the torso controller (via CAN-IR-CAN relay signals) to align and prepare docking with the other leg. The same target/threshold comparison and docking procedure as described is repeated between the torso/leg combination and other leg cluster. Picture **c** shows the system approaching last docking state.

When the torso controller has confirmed both leg cluster connections, it is ready to command the system to stand upright and walk again. At first, the accelerometer will report that the system is lying flat on the ground (pitch between -45° and 45°) and the controller will command the system to carry out its standing up gait (picture **d**). It continues this until the gait is complete and torso has confirmed that it is upright (pitch between 45° and 135°). The system is now in its original configuration and reenters its original walking routine (picture **e**). The reassembly cycle is complete and prepared for another explosive event.

Control for the corresponding leg controllers is similar to the above description with the key difference that during the connected states, it receives instead of sends gait step commands for locomotion (the torso acts as the coordinator).

For this iteration of the reassembly experiments (recent developments are described the following section), approximately 40 trials were conducted with 7 fully successful sequences. Each cycle took approximately 6-7 minutes from the initial explosive kick to full reassembly, standing and walking. Throughout the development of these trials, various system parameters were adjusted incrementally to advance the system toward greater robustness and reliability. For example, individual camera distance and angle measurements were re-calibrated to improve localization information for the position-sensitive docking phases. Various heuristic error filters were added during development such as corrections for guided locomotion during docking (i.e., if two clusters are trying to dock but pass each other, they will not detect one another; in this case reverse motions until clusters are detected again). Environment sensitive modifications were also made throughout experimental development. Examples of this include installing IR filters for the camera modules to reduce ambient IR noise and modifying the magnet faces to absorb shock (rubber inserts), guide accurate cluster docking (notch and groove features), and reduce inter-cluster IR message scattering during docking (painted surfaces).

5.5 Recent Progress and Next Steps

At the time of this writing, a few technical developments have been made and a generalizing of the reassembly algorithm is in progress. Some of the completed upgrades to the system include:

- Integration of ZigBee wireless protocol for inter-cluster and off-board PC communication;
- Centralized computation transferred from torso cluster to off-board PC;
- Cluster gait selections and synchronizations are direct from the PC to the controllers, instead of through the IR-CAN relay between Torso and Left/Right leg controllers (docking confirmation relay is still used);
- Integration of controller into Camera module (elimination of separate controller submodule unit);
- Upgraded camera hardware (faster processing chip, light filters to reduce LED noise);

- Inter-cluster magnet faces made from machined aluminum;
- Graphical visualization monitor of clusters.

The primary upgrade to the system is the migration of central computing to an off-board PC. This change simplifies the inter-cluster communication scheme, allows for effective system monitoring during runtime (helpful for software development), and reduces the computational workload of the relatively limited embedded processors. Other state machine and camera hardware/calibration refinements also contributed to a more reliable system, resulting in faster and more reliable docking (the most delicate part of the reassembly procedure). These key changes have resulted in a roughly 70% success rate for full SAE sequences.

As mentioned earlier, although the hardware of all clusters is identical, the software is not. The next step in this work is to generalize the software to be identical between clusters, allowing the robot clusters to reassign their roles (swapping of leg and torso designations) during the reassembly runtime. With the ability to relabel, more options are available for the method of cluster reassembly; labeling the clusters with respect to the center of mass (with torso in the center, left and right legs reassigned according to their positions with respect to the cluster) is a viable option. Also, six clusters have been fully constructed, allowing for larger robot structures; plans for a five-cluster quadruped built from these clusters is in progress.

Chapter 6

Conclusion

Over the past 20 years, modular robotic systems have evolved and made some impressive gains toward a goal of greater autonomy and robustness. In this dissertation, we have described our CKBot system and compared its features with the current state-of-the-art modular robots. We have presented work on isomorphic configuration recognition, distributed communication fault tolerance, and control and communication in self-reassembly for the CKBot in an effort to contribute to this push forward for modular robotics.

Using basic principles for graph theory, we have been able to apply and verify a novel method for a modular robot to recognize its shape and automatically execute an appropriate mode of locomotion. We have presented and implemented a general method for robust decision making with CKBot in the presence of faulty IR communications lines. This distributed approach, based on triple modular redundancy, is quite general and applicable to other modular systems with similar communication networks. Lastly, camera-guided self-reassembly with CKBot is a first for a modular robotic system and we have presented some of the inner workings of its closed-loop algorithm.

The important features of autonomy and robustness for modular robots are becoming increasingly relevant as mechatronic components become cheaper and smaller while researchers' access to innovative algorithms becomes more ubiquitous and abundant. Realistically, the goal of advanced modular robotic autonomy is not in the immediate future, but with each pioneering study, our understanding of this vision becomes progressively tangible. Following the trend of the past few decades, we can expect impressive and inspiring progress for these novel systems, ensuring that modular robots will be an important focus of development and research for many years to come.

Bibliography

[1] Z. Butler, R. Fitch, D. Rus, and Y. Wang. Distributed goal recognition algorithms for modular robots. In *P IEEE International Conference on Robotics and Automation (ICRA)*, pages 110–116, Washington, D.C., May 2002.

[2] 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)*, volume 1, pages 809–816, Washington, D.C., May 2002.

[3] A. Castano, W.-M. Shen, and P. Will. Conro: Towards deployable robots with inter-robots metamorphic capabilities. *Autonomous Robots*, 8(3):309–324, June 2000.

[4] A. Castano and P. Will. Representing and discovering the configuration of conro robots. In *P IEEE International Conference on Robotics and Automation (ICRA)*, pages 3503–3509, Seoul, Korea, May 2001.

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

[6] I. M. Chen and J. Burdick. Enumerating the non-isomorphic assembly configurations of a modular robotic system. *INT J ROBOT RES*, 17(7):702–719, July 1998.

[7] G. Chirikjian. Kinematics of a metamorphic robotic system. In *P IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 449–455, San Diego, CA, USA, May 1994.

[8] F. R. K. Chung. *Spectral Graph Theory.* AMS, Providence, 1997.

[9] J.-G. Dumas, C. Pernet, and Z. Wan. Efficient computation of the characteristic polynomial. In *Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 140–147, Beijing, China, 2005.

[10] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume I. John Wiley and Sons, Inc., New York, 1957.

[11] T. Fukuda and Y. Kawauchi. Cellular robotic system (cebot) as one of the realization of self-organizing intelligent universal manipulator. In *P IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 662–667, Cincinnati, OH, USA, May 1990.

[12] S. C. Goldstein and T. C. Mowry. Claytronics: A scalable basis for future robots. In *RoboSphere 2004*, Moffett Field, CA, November 2004.

[13] D. Gomez-Ibañez, E. A. Stump, B. P. Grocholsky, V. Kumar, and C. J. Taylor. The robotics bus: A local communications bus for robots. In D. W. Gage, editor, *P SOC PHOTO-OPT INST*, volume 5609 of *Mobile Robot XVII*, pages 155–163, Philadelphia, PA, December 2004.

[14] R. Groß. *Self-Assembling Robots.* Université Libre de Bruxelles, Bruxelles, Belgium, 2007.

[15] F. Harary. *Graph Theory.* Addison-Wesley, Reading, Massachusetts, 1969.

[16] L. Hogben. Spectral graph theory and the inverse eigenvalue problem of a graph. *International Linear Algebra Society*, 14:12–31, 2005.

[17] http://en.wikipedia.org/wiki. *Self-Reconfiguring Modular Robotics.* Wikipedia Foundation, 2009.

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

[19] 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*, volume 20:4, page 1730. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[20] 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*, 27(9):1069–1090, September 2008.

[21] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM T PROGR LANG SYS*, 4(3):382–401, July 1982.

[22] K. Lee, M. Moses, and G. Chirikjian. Robotic self-replication in structured environments: Physical demonstrations and complexity measures. *INT J ROBOT RES*, 27(3-4):387–401, March/April 2008.

[23] R. E. Lyons and W. Vanderkulk. Use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, April 1962.

[24] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[25] ModLab. *http://modlab.seas.upenn.edu.* University of Pennsylvania, 2009.

[26] F. Mondada, L. M. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneuborg, and M. Dorigo. The cooperation of swarm-bots: physical interactions in collective robotics. *IEEE Robotics and Automation Magazine*, 12(2):21–28, June 2005.

[27] S. Murata, K. Kakomura, and H. Kurokawa. Docking experiments of a modular robot by visual feedback. In *P IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 625–630, Beijing, China, October 2006.

[28] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *P IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 441–448, San Diego, CA, USA, May 1994.

[29] S. Murata, E. Yoshida, A. Kamimura, H. Kurokawa, K. Tomita, and S. Kokaji. M-tran: Self-reconfigurable modular robotic system. *IEEE/ASME Transactions on Mechatronics*, 7(4):431–441, December 2002.

[30] E. Østergaard. *Distributed Control of the ATRON Self-Reconfigurable Robot.* University of Southern Denmark, Odense, 2004.

[31] M. Park, S. Chitta, A. Teichman, and M. Yim. Automatic configuration recognition methods in modular robots. *INT J ROBOT RES*, 27(3-4):403–421, March/April 2008.

[32] M. Park and M. Yim. Distributed control and communication fault tolerance for the ckbot. In *ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots (ReMAR 2009)*, pages 682–688, London, UK, June 2009.

[33] PCAN-VIEW. *http://www.peak-system.com/.* PEAK-System Technik GmbH, 2009.

[34] L. S. Penrose. Self-reproducing machines. *Scientific American*, 200(6):105–114, June 1959.

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

[36] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Autonomous Robots*, 10(1):107–124, 2001.

[37] B. Salemi, M. Moll, and W.-M. Shen. Superbot: A deployable, multi-functional, and modular self-reconfigurable robotic system. In *P IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, October 2006.

[38] B. Salemi and W.-M. Shen. Distributed behavior collaboration for self-reconfigurable robots. In *P IEEE International Conference on Robotics and Automation (ICRA)*, pages 4178–4183, New Orleans, USA, April/May 2004.

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

[40] 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.

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

[42] W.-M. Shen and P. Will. Docking in self-reconfigurable robots. In *P IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1049–1054, Maui, Hawaii, USA, October 29-November 3 2001.

[43] D. A. Spielman. Faster isomorphism testing of strongly regular graphs. *STOC 96: 28th Annual ACM Symposium on Theory of Computing*, pages 576–584, 1996.

[44] K. Støy and Radhika Nagpal. Self-repair through scale independent self-reconfiguration. In *P IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2062–2067, Sendai, Japan, September 28-October 2 2004.

[45] 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)*, pages 309–316, Marina del Rey, California, March 25-27 2002.

[46] K. Støy, W.-M. Shen, and P. Will. Implementing configuration dependent gaits in a self-reconfigurable robot. In *P IEEE International Conference on Robotics and Automation (ICRA)*, pages 3828–3833, Taipei, Taiwan, 2003.

[47] I. A. Sucan, J. F. Kruse, M. Yim, and L. E. Kavraki. Kinodynamic motion planning with hardware demonstrations. In *P IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1661–1666, September 2008.

[48] C. J. Taylor and B. Shirmohammadi. Self localizing smart camera networks and their applications to 3d modeling. In *ACM SenSys/First Workshop on Distributed Smart Cameras (DSC 06)*, October 2006.

[49] J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana, IL, 1966.

[50] P. J. White and M. Yim. Scalable modular self-reconfigurable robots using external actuation. In *P IEEE International Conference on Robotics and Automation (ICRA)*, pages 2773–2778, San Diego, CA, 2007.

[51] M. Yim. *Locomotion With a Unit-Modular Reconfigurable Robot.* Stanford University, Palo Alto, CA, 1994.

[52] M. Yim. Planetary contingency: A competition educating graduate students in reconfigurable robotics. *IEEE Robotics and Automation Magazine Education Column*, 14(4):14–16, 2008.

[53] M. Yim, D. G. Duff, and K. D. Roufas. Polybot: a modular reconfigurable robot. In *P IEEE International Conference on Robotics and Automation (ICRA)*, pages 514–520, San Francisco, CA, April 2000.

[54] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian. Modular self-reconfigurable robot systems [grand challenges of robotics]. *IEEE Robotics and Automation Magazine*, 4(1):43–52, March 2007.

[55] 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)*, pages 2767–2772, San Diego, CA, October 29-November 2 2007.

[56] M. Yim, P. J. White, M. Park, and J. Sastra. Modular self-reconfigurable robots. In *Encyclopedia of Complexity and Systems Science*, pages 5618–5631. Springer New York, 2009.

[57] C.-H. Yu and R. Nagpal. Distributed consensus and self-adapting modular robots. In *IROS-2008 workshop on Self-Reconfigurable Robots and Applications*, 2008.

[58] M. M. Zavlanos and G. J. Pappas. A dynamical systems approach to weighted graph matching. *Automatica*, 44(11):2817–2824, 2008.

[59] V. Zykov, A. Chan, and H. Lipson. Molecubes: An open-source modular robotics kit. In *International Conference on Intelligent Robots and Systems (IROS) Workshop on Self-Reconfigurable Robots*, 2007.

[60] V. Zykov, E. Mytilinaios, B. Adams, and H. Lipson. Self-reproducing machines. *Nature*, 435:163–164, 2005.