9-12-2002

Heterogeneous Self-Reconfiguring Robotics: Ph.D. Thesis Proposal

Robert C. Fitch
Dartmouth College

Heterogeneous Self-Reconfiguring Robotics

Ph.D. Thesis Proposal

Robert C. Fitch
Department of Computer Science
Dartmouth College

Dartmouth Computer Science Technical Report TR2002-436

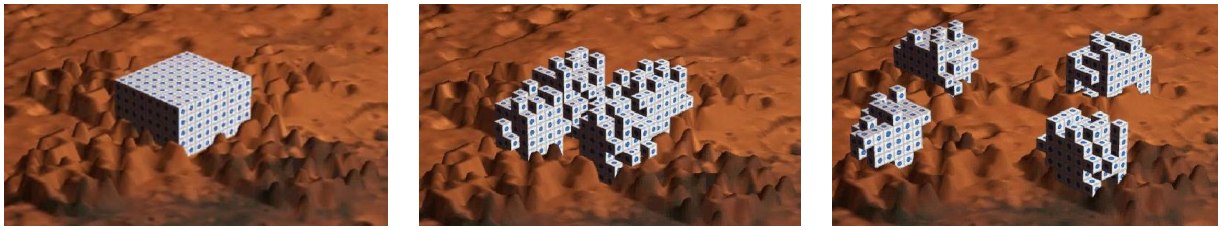
September 12, 2002

Abstract

Self-reconfiguring robots are modular systems that can change shape, or *reconfigure*, to match structure to task. They comprise many small, discrete, often identical modules that connect together and that are minimally actuated. Global shape transformation is achieved by composing local motions. Systems with a single module type, known as *homogeneous* systems, gain fault tolerance, robustness and low production cost from module interchangeability. However, we are interested in *heterogeneous* systems, which include multiple types of modules such as those with sensors, batteries or wheels. We believe that heterogeneous systems offer the same benefits as homogeneous systems with the added ability to match not only structure to task, but also capability to task.

Although significant results have been achieved in understanding homogeneous systems, research in heterogeneous systems is challenging as key algorithmic issues remain unexplored. We propose in this thesis to investigate questions in four main areas: 1) how to classify heterogeneous systems, 2) how to develop efficient heterogeneous reconfiguration algorithms with desired characteristics, 3) how to characterize the complexity of key algorithmic problems, and 4) how to apply these heterogeneous algorithms to perform useful new tasks in simulation and in the physical world. Our goal is to develop an algorithmic basis for heterogeneous systems. This has theoretical significance in that it addresses a major open problem in the field, and practical significance in providing self-reconfiguring robots with increased capabilities.

Contents



(a) (b) (c)

Figure 1: Simulation of SR robot on Mars terrain. The robot begins as a single cube in (a), but divides into four in (b) and (c) for parallel exploration using a distributed, waterfall-like locomotion algorithm inspired by cellular-automata.

1 Introduction

Self-reconfiguring (SR) robots are robots that can change shape to match the task at hand. These robots comprise many discrete modules, often all identical, with simple functionality such as connecting to neighbors, limited actuation, computation, communication and power. Orchestrating the behavior of the individual modules allows the robot to approximate, and reconfigure between, arbitrary shapes. This shape-changing ability allows SR robots to respond to unpredictable environments better than fixed architecture robots. Common examples of reconfigurability in action include transforming between snake shapes for moving through holes and legged locomotion for traversing rough terrain, and using reconfiguration for locomotion as shown in Figure 1. SR robots also show promise for a high degree of fault tolerance since modules are generally interchangeable. If one module fails, the robot can self-repair by replacing the broken unit with a spare stored in its structure. When all modules are the same, the system is known as *homogeneous*. This design constraint promotes fault-tolerance and versatility. However, a homogeneous system has limitations; all resources that may be required must be built into the basic module. We would like to relax the assumption that all modules are identical and investigate *heterogeneous* systems, where several classes of modules work together in a single robot. We believe that heterogeneous systems can retain the advantages of their homogeneous counterparts while offering increased capabilities. The benefit would be a robot that can match not only *structure* to task by changing physical configuration, but also *capability* to task by using specialized components.

Consider a future application in which exploration tasks are carried out by SR robots. When necessary, the robot reconfigures into a legged walker to move across rough terrain or rubble, or transforms into a snake shape for moving through small holes. The robot can take advantage of smooth terrain as well by deploying a special module type containing wheels for fast, efficient locomotion. A variety of sensors are onboard, contained as modules within the structure of the robot. The sensor modules are surrounded by other modules for protection, but reconfigure to the surface when needed for better performance. Power is provided by dedicated battery modules also stored in the structure of the robot. Since they are relatively heavy, these non-actuated battery modules remain close to the base of the robot during reconfigurations, but are maneuvered closer to modules that draw a large amount of current. As particular batteries are drained, others are positioned to take their place. Long range communication with human users is accomplished through a small number of radio modules. Communications algorithms work for any number of radios to provide fault tolerance against single radio failure. This property, that performance degrades gracefully in response to module failure, is true of most of the robot's resources. For example, if enough wheel modules fail then the robot locomotes using legs. If further failure occurs the robot employs an inchworm gait requiring very few operational actuators.

Another example application is self-assembly. Large SR modules of varying sizes and shapes are deployed in small groups to a remote construction site, where they rapidly self-assemble into scaffolding, buildings or temporary structures. This construction technique is utilized at Martian outposts or other sites that are difficult to reach. If a building is no longer needed, it simply reconfigures into a different structure. Alternatively, SR robots can form a kind of reconfigurable factory. An assembly task is divided into a number of subassembly tasks, which are completed by a SR robot configured specifically for that subassembly. There are specialized components for peg-in-hole type assemblies, gluing, welding or other tasks. The SR robot is simple and task-specific for the given subassembly, yet can accommodate a variety of different subassemblies through reconfiguration.

The robots described in these scenarios are heterogeneous. Most research to date, however, focuses on homogeneous systems[1]. Contributions are numerous and many algorithmic and hardware challenges have been met. But, the natural limitations of homogeneous systems lead us to investigate heterogeneity. For example, it would be difficult to argue for building a robot with 1000 radio transmitters, 1000 deployable wheels, and 1000 infrared cameras, as would be required by a homogeneous system, when significantly fewer resources are sufficient. One potential solution is to construct complex modules from very small homogeneous parts, but in this case the granularity of module size must be extremely high versus size of the overall robot. Even at MEMS-scale, real estate is limited.

Castano and Will [13] compare homogeneous and heterogeneous designs in terms of a trade-off between software complexity and hardware complexity. They observe that a common module design benefits from economies of scale in design and manufacturing, and from simpler software requirements. Choosing multiple module types increases the cost of both hardware and software production. However, as desired capabilities increase, the complexity and cost of the base module also increases. We wish to examine tradeoffs between the number of module types in a robot and the complexity of the basic module in a task-directed way. We believe that heterogeneous systems can benefit from both the redundancy of homogeneity and the efficiency of specialization.

Another argument in favor of considering heterogeneous systems is that adding seemingly trivial heterogeneity from a hardware perspective incapacitates homogeneous reconfiguration algorithms, which cannot distinguish a specialized module from any other. It is impossible to, for example, maintain a particular sensor module on the surface of the robot after reconfiguration. If any module differences, however small, are desired, a heterogeneous reconfiguration algorithm is required.

1.1 Challenges

There are significant challenges involved in designing heterogeneous SR systems. A fundamental issue is the degree to which modules are different from each other. There are many possible dimensions of heterogeneity, such as size and shape differences, various sensor payloads, or different actuation capabilities. These differences all impact the main algorithmic problem, which is how to reconfigure when all units are not identical. Current reconfiguration algorithms are based on homogeneity. Heterogeneous reconfiguration planning is similar to the *Warehouse problem*, which is $PSPACE-$hard in the general case [71, 32]. Beyond the reconfiguration planning problem itself, many other challenges remain in developing applications that capitalize on module specialization.

In response to these challenges, we would like to develop an algorithmic basis for heterogeneous self-reconfiguring robots, and to develop software simulations that demonstrate our solutions along with hardware experiments where possible. To this end we propose four main research questions:

1. **Framework for heterogeneity.** There are many possible differences between SR modules. In order to reason about heterogeneous systems, a categorization scheme is required that models the various dimensions

[1] It is interesting to note, though, that the original vision of SR robots, outlined by Fukuda in 1987 [28] describes a heterogeneous system with dedicated wheel modules and joint modules, among others.

of heterogeneity. The benefit of such a framework is that algorithms can be developed for classes of systems instead of specific robots. We will identify some primary axes of heterogeneity and build this framework.

2. **Reconfiguration algorithms.** Reconfiguration planning is the main algorithmic problem in SR systems. Since homogeneous reconfiguration algorithms are insensitive to module differences, we need to develop a new class of reconfiguration algorithms that are distributed, scalable, and take into account different types of resources, or tradeoffs between resources. Our approach to the problem is based on a special case of the Warehouse problem, which admits polynomial-time solutions in contrast with the intractability of the general case.

3. **Lower bounds for reconfiguration.** We propose to determine lower bounds for the complexity of reconfiguration problems under various assumptions about heterogeneity as developed in our framework defined above. We would like to complete such analyses for as many categories as possible.

4. **Applications.** Although a solution to the heterogeneous reconfiguration problem is significant from a theoretical perspective, we are also interested in developing example applications in simulation and in the physical world. For example, we would like to add specialized, non-actuated battery modules to a system. Neighbor units would need to cooperate to move the batteries, but this application can allow systems which currently use off-board power to operate untethered. Another example is deploying specialized modules with grippers for assembly tasks.

1.2 Outline

This proposal is organized as follows. Chapter 2 discusses previous work in self-reconfiguring robotics and related areas. In Chapter 3, we detail major research issues in heterogeneous SR systems. Chapter 4 describes our proposed research. Previous research we have completed in SR robotics is described in Chapter 5, and Chapter 6 concludes with a summary of the proposal, expected contributions, and a research timeline.

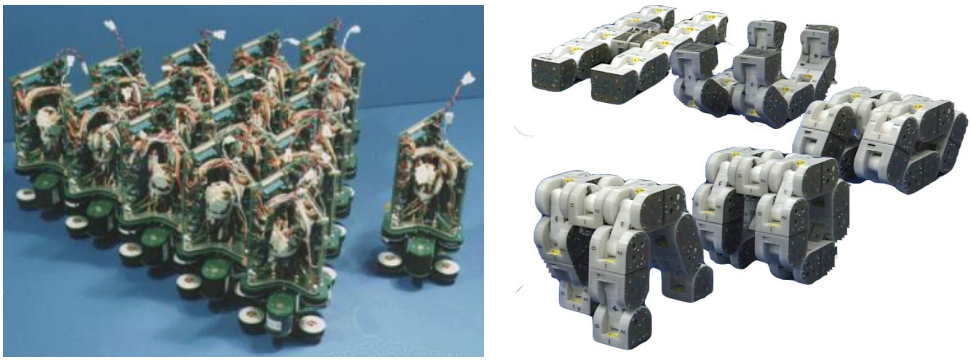
2 Related Work

Research in modular and self-reconfiguring robots spans 15 years [28]. During this time, researchers have obtained significant theoretical and practical results, and hardware prototypes have progressed from 2D tethered units such as the Fracta (Figure 2, left) to sleeker, 3D robots such as MTRAN (Figure 2, right). Work relevant to this proposal falls into the categories of hardware design, algorithms and theory, and cooperative robotics.

2.1 Hardware Design

Building reconfiguring robots in hardware involves designing and constructing the basic modular units that combine to form the robot itself. Such modules differ from the wheels, arms, and grippers of fixed architecture robots in that they are functional only as a group as opposed to individually. Because we are interested in developing general algorithms for classes of robots instead of particular systems, familiarity with the entire spectrum of existing systems is valuable. Current systems can be divided into classes based on a number of module properties. Systems composed of a single module type are known as homogeneous systems, and those with multiple module types are called heterogeneous systems. Modules can connect to each other in various ways, and the combination of connection and actuation mechanisms determine the possible positions a module can occupy in physical space, relative to neighbor modules. This gives rise to the major division within SR systems, *lattice-based* versus *chain-based* systems. In lattice-based systems, modules move among discrete positions, as if embedded in a lattice. Chain-based systems, however, attach together using hinge-like joints and permit snake-type configurations that connect to form shapes such as

legged walkers and tank treads. See Figure 3 for typical examples from each class. Another class of modular systems cannot self-reconfigure, but can reconfigure with outside intervention. This class is called *manually reconfiguring* systems. In this section, we survey robots in each of these categories.



(a) (b)

Figure 2: Examples of self-reconfigurable robots. The *Fracta* robot of Murata, Kurokawa and Kokaji [45] is shown in (a), tethers unplugged. In (b), the newer *MTRAN* robot of Murata et al. [47] is shown reconfiguring into a legged walker.

2.1.1 Pioneering Research

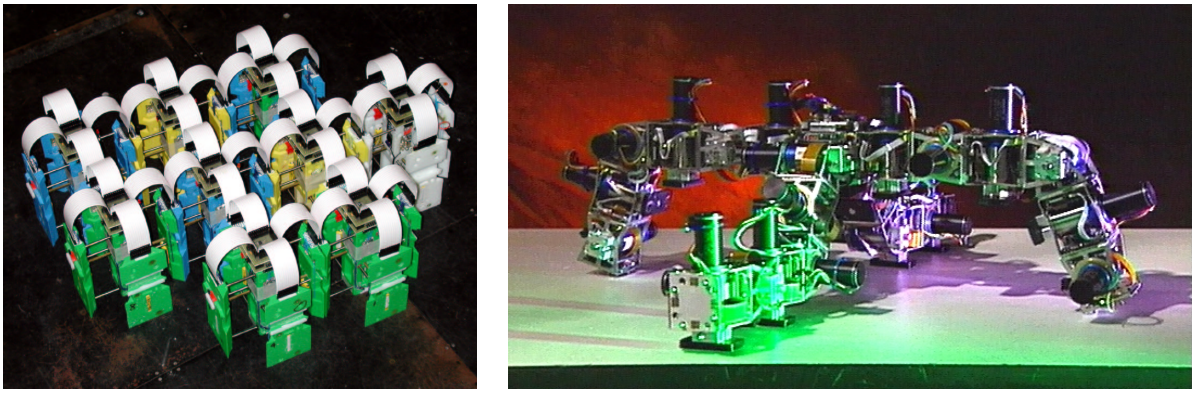
CEBOT (cell structured robot) was the first proposed self-reconfiguring robot, introduced by Fukuda et al. in 1988 as an implementation of their 1987 idea of a *Dynamically Reconfigurable Robotic System* (DRRS) [29, 28]. The definition of DRRS parallels our current conception of self-reconfiguring robots - the system is made up of robotic modules (cells) which can attach and detach from each other autonomously to optimize their structure for a given task. The idea is directly inspired by biological concepts and this is reflected in the chosen terminology. It is interesting that this proposed SR robot is heterogeneous: cells have a specialized mechanical function and fall into one of three "levels". Level one cells are joints (bending, rotation, sliding) or mobile cells (wheels or legs). Linkage cells are part of Level two, and Level three contains end-effectors such as special tools. Communication and computation are assumed for all cells.

CEBOT is the physical instantiation of DRRS. Various versions range from reconfigurable modules [27] to "Mark-V," which more closely resembles a mobile robot team [10].

2.1.2 Lattice-based Robots

In lattice-based systems, modules are constrained to occupy positions in a virtual grid, or lattice. One of the simplest module shapes in a 2D lattice based system is a square, but more complex polygons such as a hexagon (and a rhombic dodecahedron in 3D) have also been proposed. Because of the discrete, regular nature of their structure, developing algorithms for lattice-based systems is often easier than for other systems. However, the grid constraint makes implementing certain rolling motions, such as the tank-tread, more challenging since module attachment and detachment is required. We would like to develop algorithms implementable by most, or all, lattice-based systems, so a complete review of their properties is essential.

One of the first lattice-based SR robots proposed and constructed in hardware is the *Fracta* robot [45] (Figure 2, left). The homogeneous, 2D *Fractum* modules connect to each other using electromagnets. Communication is achieved through infrared devices embedded in the sides of the units, and allows one fractum



(a) (b)

Figure 3: Examples of (a) a lattice-based system, the Crystal, and (b) a chain-based system, Polybot [74].

to communicate with its neighbors. Computation is also onboard; each fractum contains an 8-bit microprocessor. Power, however, is provided either through tethers or from electrical contacts with the base plane. This system was designed for self-assembly, and can form simple symmetric shapes such as a triangle, as well as arbitrary shapes [67]. Other lattice-based robots from the same group include a smaller 2D system [77], and a 3D system [46].

Another early SR robot is the *Metamorphic Robot*[52]. The basic unit of this robot is a six-bar linkage forming a hexagon. The kinematics of this shape were investigated when the design was proposed [16], and hardware prototypes were constructed later [52]. A unique characteristic of this system is that it can directly implement a convex transition; a given module can move around its neighbor with no supporting structure. The hexagon deforms and translates in a flowing motion. A square shape with this same property was also proposed. This motion primitive is important since it is required by many general reconfiguration algorithms, but many systems can only implement it using a group of basic units working together.

The Metamorphic Robot module was a hexagon in two dimensions, but a similar 3D module, the *Rhombic Dodecahedron*, was later proposed by Yim et al. [75]. The Rhombic Dodecahedron has 12 faces, and each face is a rhombus. Rhombic Dodecahedron modules pack tightly in a 3D grid and locomote by rolling around each other. This shape is an example of a class called *Proteo* modules [76].

The first module proposed by the Dartmouth group is the *Molecule* [40]. This 3D module consists of two *atom* units connected by a 90-degree *bond*, forming the overall shape of an elbow with a connection mechanism at each end. The molecule is capable of both convex and concave transitions using its rotation-based actuation mechanism [38].

A variation on this actuation scheme is AIST's *Modular Transformer* (MTRAN) [47], shown in Figure 2, right. Like the Molecule, the MTRAN design is also based on two components connected by a link. The difference is that MTRAN's semi-cylindrical end components each can rotate 180 degrees, resulting in a variable bond angle. This actuation allows MTRAN to behave as both a lattice-based and a chain-based system. The modules can be closely packed in a 3D grid, or they can form chains such as legs in a legged-walker configuration. Communication is onboard the MTRAN, and power from an external source is transmitted between units through connections on the faces. Experiments demonstrating various modes of locomotion, including crawling and quadruped gaits, have been performed, as well as reconfiguration between modes.

Another variation on this design is the *I(CES)-Cubes* robot [69] . The link component on this module, however, is separated from the end components. Therefore, the system is termed *bipartite*. The end units, the *cubes*, are passive and connect to *Link* units, which are actuated. In this way, the I(CES)-cubes implement

motion primitives similar to the Molecule robot [68], although there are fewer constraints since each cube can be placed independently.

Motion of the previously discussed modules is primarily achieved by movement over the surface of the robot. A separate class, *unit-compressible* systems, use modules which move through the volume of the robot. The actuation method of unit-compressible modules is termed *scaling-based* because the modules expand and contract in multiple dimensions. The basic idea of using extendable arms for actuation to construct a reconfigurable robot was patented by Tanie and Maekawa in 1993 [66]. The *Crystal* robot was constructed by the Dartmouth group as a 2D physical realization of a unit-compressible system [71, 56, 57]. Crystal units are squares that attach to each other at each of the four faces, and expand and contract in two dimensions. Communication in the Crystal robot is between neighbor units only, and is implemented using infrared devices mounted in the faces. Both power and computation for the Crystal are onboard each unit, distinguishing this robot as one of the few untethered SR systems.

A unit-compressible system extended to three dimensions was developed at Xerox PARC [41]. This implementation, the *Telecube*, is similar to the Crystal but has an added degree of expansion and contraction for the third dimension [65]. An external power source is required but the Telecube avoids a large number of tethers by routing power between modules.

Other actuation models have also been proposed for lattice-based systems. Hosokawa et al. [33] designed and constructed a system based on cubes with actuated arms. The arms attach to other units and allow one cube to pull another cube from its side to its top. This robot is two-dimensional, but in the vertical plane. A system using pneumatic actuators, along with hardware experiments, is presented by Inou, Koseki and Kobayashi [35].

2.1.3 Chain-based Robots

In chain-based systems, modules aggregate as connected 1D strings of units. This class of robots easily implements rolling or undulating motions as in snake robots or legged robots. However, control is much more difficult for chain-based systems than for lattice-based systems because of the continuous nature of the actuation: modules can move to any arbitrary position as opposed to a fixed number of neighbor positions in a lattice. Our previous work has not considered chain-based systems, but it is important to understand their characteristics in the interest of developing more generalized algorithms.

The first prominent chain-based system was *Polypod*, proposed by Yim in 1993 [72, 73]. Polypod is made up of *Segments*, which are actuated 2-DOF 10-bar linkages, and *Nodes*, which are rigid cubes housing batteries. Multiple gaits for locomotion, including rolling, legged, and even Moonwalk gaits, were demonstrated with Polypod. *Polybot* succeeds Polypod, sharing the same bipartite structure. Segments in Polybot abandon the 10-bar linkage in favor of a 1-DOF rotational actuator. The latest generation of Polybot prototypes has on-board processing and CANbus (controller area network) hardware for communication.

A system that uses a similar actuation design is CONRO (CONfigurable RObot) [63]. The CONRO module has two rotational degrees of freedom, one for pitch and one for yaw, and was designed with particular size and weight considerations [12]. Considerable attention has been paid to the connection mechanism, which is a peg-in-hole connector with SMA (shape-memory alloy) latching mechanism that can be disconnected by either face. Computation is on-board each module, so unlike Polypod, CONRO has only one module type. Power can be provided externally or via batteries on later prototypes [13]. Examples of manually configured shapes are the snake and the hexapod, and the current CONRO system is designed for self-reconfiguration.

The DRAGON is a snake robot with torsion-free (constant-velocity) joints [49]. A sophisticated connector has been developed for the DRAGON, designed for strength[2] and tolerance for docking [50].

[2]The author demonstrated this by suspending himself from the connector, hanging from the sixth floor of a building.

2.1.4 Manually Reconfigurable Robots

Modularity, the concept of designing a system divisible into discrete and relatively independent pieces, is often considered a desirable characteristic in designing systems. Manually reconfigurable modular systems share many design issues with self-reconfigurable systems, so a brief review is warranted. Many groups have developed modular robotics [18, 44] and manipulators, such as the *Reconfigurable Modular Manipulator System* (RMMS) of Paredis, Brown and Khosla [54], and Goldenbergs modular arm [34]. The general problem of reconfigurable modular design is explored by Chen and Burdick [14], and Farritor and Dubowski [20].

2.1.5 Heterogeneous Systems

SR systems with significant levels of heterogeneity are few. The most prominent heterogeneous system is CEBOT [28], although bipartite systems such as I(CES)-Cubes and Polybot are minimally heterogeneous. Even though few existing robots are significantly heterogeneous, it is nonetheless beneficial to develop heterogeneous algorithms as module differences do arise in practice. Heterogeneity results from adding various sensors to any existing system, for example. Another source of heterogeneity is module failure; failed actuators lead to a system with immobile modules. Differential battery drain in systems with on-board power can cause actuation speed differences between modules. This heterogeneity might seem trivial from a mechanical perspective but causes major problems algorithmically since it leads to violations of the assumption of module interchangeability.

2.2 Planning and Control

In attempting to devise new algorithms, it is useful to understand techniques developed previously. In this section, we survey early theoretical work, approaches to the reconfiguration problem, and also work addressing the locomotion and self-repair problems.

2.2.1 Early Algorithms

In SR research, CEBOT work is prescient in that a majority of current research issues in SR robotics are identified in early CEBOT papers. Communication and docking problems are described in [25]. Distributed decision making [26], hierarchical control [30, 10] and analysis of how the number of modules affects performance [36] are also studied. Other early theoretical work in planning and control for modular systems comes from the perspective of cellular automata theory. Gerardo Beni proposed the idea of a "cellular robotic system" around the same time as the first CEBOT papers [2]. Here the familiar idea of large numbers of mobile cooperating autonomous units is reiterated, but physical reconfiguration is not included. Distributed control with no synchronized clock is emphasized. Further papers discuss theoretical [3] and practical [31] issues. For additional information on early cellular and cooperative robotics research, see survey papers by Sandini [60] and Cao, Fukunaga and Kahng [11].

2.2.2 Reconfiguration Planning

The task of transforming a modular system from one configuration into another is called the *Reconfiguration Planning* problem. Solving this problem is fundamental to any SR system. In some approaches explicit start and goal configurations are given, and in others the goal shape is defined by desired properties. *Centralized* algorithms require global system knowledge and compute reconfiguration plans directly, whereas *decentralized* algorithms compute solutions in a distributed fashion without the use of a central controller.

Reconfiguration algorithms can be designed for specific robots, or for classes of modules. Often a centralized solution is more obvious and is developed first, followed by a distributed version, although not always. Not all decentralized algorithms are guaranteed to converge to a solution, or are correct for arbitrary goal shapes. We review relevant reconfiguration algorithms in this section.

CEBOT reconfiguration was planned by a central control cell known as a *master* [27]. Master cells were later intended to be dynamically chosen, blurring the distinction between centralization and decentralization [30]. Later CEBOT control is hierarchical (behavior-based) [10].

A common technique used in reconfiguration algorithms for lattice-based systems is to build a graph representation of the robot configuration, and then to use standard graph techniques such as search to compute motion plans. Planning for the Molecule robot developed by the Dartmouth group is one example [38, 39]. Another example from the Dartmouth group is planning for unit-compressible systems such as the Crystal [71, 56]. This planner, named *MeltGrow*, uses the concept of a *metamodule*, where a group of modules are treated as a single unit with additional motion capabilities. The Crystal robot implements convex transitions using metamodules called *Grains*. Graph-based algorithms are also used by the MTRAN planner to compute individual module trajectories [78].

Centralized planners can also store pre-computed data structures such as gait-control tables. Once a gait is selected by the central controller, it is executed by local controllers on the individual modules. This type of algorithm is used by Polypod [73]. The division between central and local controllers is also used in by RMMS [54], and I-Cubes [69].

Important work in decentralized planning begins with the Fracta system [45]. Individual units used a precompiled set of rules to self-assemble into various shapes, and eventually into arbitrary shapes [67]. The randomized component of these algorithms limits convergence guarantees, however, as the algorithms are similar to simulated annealing. A similar algorithm is presented by Hosokawa et al. [33]. This algorithm uses simpler rules and is deterministic, but is more limited in the classes of shapes it can form. Shapes with "overhangs" are disallowed, for example. Yim, Duff and Roufas [76] present a distributed controller for Proteo modules that achieves arbitrary shapes, but again without convergence guarantees.

A successful approach in distributed planners is the use of message-passing. Shen, Salemi and Will [62] and Salemi, Shen and Will [59] propose a control system for CONRO using a message-passing scheme called *Digital Hormones*. The problem of distributed reconfiguration for unit-compressible modules was solved by a combination of the *Pacman* algorithm developed by the Dartmouth group [4] and later modifications and analysis by Vassilvitskii, Yim and Suh [70]. This distributed algorithm is correct and complete for arbitrary shape reconfigurations of 3D unit-compressible cubic systems using metamodules, but makes explicit use of module homogeneity to achieve efficiency.

In addition to algorithmic solutions, other theoretical issues related to the reconfiguration problem have been addressed. Chirikjian and Pamecha [17] discuss upper bounds for self-reconfiguration. Metrics for reconfiguration planning have also been studied [53, 15], and Vassilvitskii, Yim and Suh provide complexity analysis for their distributed reconfiguration algorithm [70]. No lower bounds analysis has been published.

2.2.3 Locomotion and Self-Repair

Aside from the reconfiguration problem, the *locomotion* problem for reconfigurable robots has also received research attention. Generally, chain-based systems locomote without module detachment, while lattice-based systems require reconfiguration to perform locomotion. Yim [73] demonstrates control for rolling and legged locomotion gaits for Polypod. Støy, Shen and Will [64] present distributed locomotion algorithms for CONRO. In our previous work, we studied distributed inchworm-style locomotion for unit-compressible systems [6]. Other distributed locomotion work, based on Cellular Automata-style algorithms, is presented by Butler, Kotay, Rus and Tomita [7].

When modules in a SR system fail, it is desirable for the robot to fix itself, or *self-repair*. The self-repair

problem was studied by Yoshida et al. [79]. We also developed self-repair algorithms in our previous work [24], proposing a three-step process for self-repair: detect, eject and replace. After a failure is detected, the failed module is ejected and replaced using path-planning minimizing number of bends and manhattan distance in two dimensions. For the 3D self-repair problem, we developed a new bend-minimizing shortest-path algorithm [23].

2.3 Other Related Results

Results from communities other than modular robotics relate to our algorithmic questions as well. This work comes from the fields of heterogeneous robot teams, self-assembly and formation control.

There is a large body of literature investigating teams of mobile robots. Often, researchers are interested in how teams of varying types of robots, or heterogeneous teams, can cooperate in accomplishing a common task. Self-reconfiguring and modular robots can be thought of as a special case of a tightly coupled robot team. See Balch and Parker [9] for an excellent review of the field.

Another related field of research is theoretical self-assembly. An interesting recent result is by Nagpal [48], who presents a programming language for constructing global shapes from independent agents using "Origami Mathematics." Other work studies how free-floating square tiles can self-assemble into global shapes [55, 1]. Inspired by biological processes, Saitou and Jakiela [58] investigate how to self-assemble subassemblies with conformational switches.

Vehicle formations are assemblies of self-controlled mobile vehicles, such as a squadron of aerial robots. The controls community has recently investigated distributed controllers for such formations. This is similar to modular robot control, without the physical coupling of modules. Representative work is presented by Fax and Murray [22, 21], Klavins [37], and Olfati-Saber and Murray [51].

3 Research Issues

Researchers in SR robotics have made significant progress in designing, building, and understanding how to control homogeneous systems. In particular, the reconfiguration problem has been addressed in 2D and 3D, and with centralized and decentralized approaches [56, 4]. Although no lower bounds analysis has been published, fast algorithms have been developed with low polynomial running times: for example, $O(n^2)$ moves for a unit-compressible system with n units [70]. These assume reconfigurable modules with simple actuation models, such as the unit-compressible model of the Crystal [57] and Telecube [70], or else sliding block motion as in Cellular Automata style algorithms [8], which are applicable to many different hardware types and provide provably correct global behavior from local actions for specific tasks. Important research issues remain, however, especially with regards to correctness, efficiency, global solution guarantees and versatility of reconfiguration algorithms. Accommodating heterogeneity is a clear example of this.

We seek algorithmic solutions that accommodate heterogeneous systems, are able to be instantiated to an array of hardware types, and are scalable and parallel. The challenge is that homogeneous algorithms are based on the idea of interchangeability of all modules, and this is no longer true of heterogeneous systems. A further challenge is that the concept of heterogeneity is ill-defined; modules can differ in a variety of ways. Suitable models that encompass all the degrees of heterogeneity need to be developed, followed by algorithms to perform basic tasks assuming certain model parameters. A system could consist of a fixed number of module classes, with interchangeability within classes, and algorithms should make use of this property, for example. Finally, these algorithms should be implemented in simulation and in hardware.

The significance of these research questions is both theoretical and practical. A proven polynomial-time solution to heterogeneous reconfiguration planning would be significant since it is an open problem that is important to the field. General algorithms of the type we envision would be applicable to a large number

of systems which currently require specialized planning and control, and would add another dimension to our understanding of reconfigurable systems. Practically, these results would enable interesting future applications of SR systems. In the near term, current systems are intended to gain functionality from large numbers of modules, but the number of units in most real systems is actually very small. Heterogeneous algorithms could allow more functionality with smaller numbers of units in the meantime as larger systems are constructed.

From these observations, we propose four classes of research problems. The root is to develop a suitable theoretical model. Next is the main problem, the reconfiguration problem, along with implementations. Various application problems and implementations then follow. These research issues are discussed in detail in the following sections. We have already completed some preliminary work in applications and implementation, described in Chapter 5. Future problems to be addressed as part of this proposal are given in Chapter 4. We organize our research agenda into four main questions:

1. How can we classify heterogeneous systems?
2. How can we develop algorithms for reconfiguration, including cases of several module modes, and for issues of resource tradeoffs during reconfiguration?
3. How can we characterize the complexity of reconfiguration planning and other capabilities?
4. How can we develop example applications both in simulation and the real world?

3.1 Framework for Heterogeneity

The majority of current SR systems can be classified by characteristics of the single modular type. They are generally categorized as lattice- or chain-based according to their method of actuation. Heterogeneous systems, however, comprise characteristics not only of the basic module types, but also the relationships and differences between module classes. Castano and Will [13] describe these differences in terms of "Levels of Homogeneity," and measure the similarity between modules as a factor in determining cost of hardware versus software. We reverse the term and wish to evaluate the "Degree of Heterogeneity" of a system. For example, a homogeneous system with wireless communication added to one module is at one end of the spectrum, and CEBOT-style systems with dedicated wheel, CPU and joint modules are at the other. The question is how to describe and measure the dimensions and degree of heterogeneity in a SR system.

Building a framework is important for understanding and comparing SR systems. Basic defining properties of modules determine what classes of algorithms can be applied to which systems. A reconfiguration algorithm for unit-compressible systems such as the Crystal works differently than reconfiguring a system such as the Molecule. However, the concept of a metamodule, defined earlier as a group of homogeneous units acting as a group, allows the same algorithm to run on both systems. This is the type of generalization we would like to apply to heterogeneous systems.

The challenge in building such models is encompassing the dimensions of difference in our framework. We must model actuation type, shape, size, communications model, capabilities, etc. such that we can build algorithms for tasks that make use of these properties. Like any model, we must be true enough to the physical system such the resulting algorithms run in hardware, while still retaining enough simplicity that we are able to analyze the complexity and prove correctness of our algorithms.

Categorizing the levels of heterogeneity and building a working model of module types is fundamental to heterogeneous SR research, and is assumed as a prerequisite for any type of algorithm development. We have developed a preliminary version of this model and propose to refine this as necessary during the remaining stages of research. The current model is described in Section 4.

3.2 Reconfiguration Planning

The reconfiguration problem is to compute a motion plan that transforms the robot from one given shape into another. This is the most basic issue for any SR system, and is an active area of research. Many algorithms have been presented for various homogeneous systems, with both centralized and distributed versions available. These solutions often rely on module homogeneity as a means for reducing complexity. Since all modules are interchangeable, techniques such as "virtual module relocation" are possible. For example Butler, Byrnes and Rus's distributed PacMan algorithm [4] (see Section 2) virtually relocates units by means of a token, or "pellet," that is passed between units to denote the symbolic relocation of a given module. Using these methods, homogeneous reconfiguration algorithms run quickly: quadratic time is the norm for unit-compressible systems. Assigning unique identifiers to modules, as necessary to support heterogeneity, was presumed by researchers to deem the problem intractable as an instance of the $PSPACE-$hard Warehouse Problem. However, a paper by Sharma and Aloimonos [61] shows how the amount of free space available in the problem can in fact reduce the running time to $O(n^2)$. This solution applies to rectangular objects even with arbitrary size ratios. The question is how to adapt such a solution to the heterogeneous reconfiguration problem for various module types.

We therefore wish to develop reconfiguration algorithms that accommodate heterogeneous systems of various hardware types. These algorithms should be scalable, parallel and easy to use. They should also be provable correct, efficient, and should offer global performance guarantees.

From this goal, several concrete issues arise. The first is how to allow unique module identifiers. Approaches based on solutions to the Warehouse problem, discussed earlier, are promising.

Because we want our algorithms to eventually be decentralized, we will encounter the problem of recognizing global properties using local information. We call one such problem the *Goal Recognition* problem, which is how to determine if the robots current global configuration matches a given target configuration. We have developed a solution to this problem previously that easily extends to the heterogeneous case.

Not all units in a heterogeneous system are necessarily actuated. Therefore we must account for the case where neighbor modules must cooperate to move non-actuated units. We have begun to explore this issue in terms of a different task known as *self-repair*, where the system cooperatively ejects a failed (therefore assumed to be non-actuated) unit. However new cooperative gaits for manipulating non-actuated units need to be developed and incorporated into reconfiguration algorithms.

Another issue that arises is how to handle resource trade-offs. An example is the case where module motion is easy in straight lines, but difficult in turns. Here it is useful to spend computation time in computing bend-minimizing paths in return for a speedup in execution time. We have previously developed a turn-minimizing shortest path algorithm for this purpose.

Specific issues we have addressed or propose to address are listed as follows:

- How to develop a reconfiguration algorithm for modules with unique identifiers;
- How to address the Goal Recognition problem;
- How to reconfigure with non-actuated modules;
- How to balance resource trade-offs.

3.3 Lower bounds for reconfiguration

A natural theoretical question that arises is, what are the computational lower bounds for reconfiguration? Determining such bounds would be a great step towards a general "Computational Theory of Self-Reconfiguration." We would like to compute lower bounds for as many classes of modules as possible. So-

lutions to the heterogeneity framework question proposed above will define such classes. We are currently developing such a model, described in a later section (see Section 4).

3.4 Applications and Implementation

After the basic reconfiguration problem is solved, the next step is to investigate the use of reconfiguration in other algorithmic applications. One such class of algorithmic questions deals with resource utilization. Heterogeneous systems allow specialized modules for power, communications, computation, mobility, or other resources. How these resources should best be distributed for various tasks is an interesting problem. For example, in a manipulation task it may be desirable to move a dedicated power module close to the task through reconfiguration. Another example is sensor deployment. Sensor modules should be carried in the volume of the robot for locomotion, and deployed to the surface for use. A related task would be to store wheel modules in the body of a legged configuration, and to deploy the wheels when wheeled locomotion was possible. Assuming a solution to the problem of reconfiguration with uniquely identified modules, the application-level question is how to best use this capability. Specifically, the research issue is to determine a target configuration that optimizes placement of power, sensor, or other specialized modules to best suit the task.

Another application involves the problem of constructing rigid structures using SR modules. Often a SR robot requires structural rigidity, but it is difficult to construct connectors with desirable connection and disconnection properties that can withstand much torque. Power and weight available to a module are both severely limited, so connectors must use small efficient actuators. The result is that current connectors have serious problems with rigidity. A line of Crystal modules, for example, can deform to a great degree. This deflection is currently unable to be controlled. Heterogeneity could possibly help this situation using various module shapes to create bracing, or a brick-like pattern. This would allow rigidity where necessary. On a larger scale, this could enable self-assembling structures such as scaffolding or even temporary buildings using SR systems. We pose the overall question as, how to build strong global structures using weak local connectors.

Any algorithms we design should be implemented and simulated in software. The challenge for heterogeneous systems is to build simulators to represent the varieties of modules. In hardware, building a heterogeneous system by adding sensors or communication to a homogeneous system is an easy strategy. It would also be interesting to construct modules of different shapes. Demonstrating general reconfiguration in hardware remains a significant goal. Overall, the research goal here is to build a suitable software simulator to test our algorithms, and to perform hardware experiments where possible.

4 Current and Proposed Research

In the previous chapter we outlined our research agenda. We present our current and proposed work in this chapter, along with our approaches to solving the open problems. This work falls into four categories: our proposed module taxonomy and theoretical model, the reconfiguration problem with variations, lower bounds analysis, and applications and implementation.

4.1 A Taxonomy for Heterogeneity

In order to better understand the various types of heterogeneous systems, we propose a basic categorical framework in Table 1. The general modes of module actuation are listed on the horizontal axis, and various types of relative shape ratios between modules are listed on the vertical axis. Actuation mode (scaling, rotation, translation, deformation) is the primary determining factor in reconfiguration. The Crystal and Telecube are scaling modules, while most other existing modules use rotation. Translation is implemented

	Scaling (Crystal)	Rotation (MTRAN, Molecule)	Translation	Deformation (Metamorphic)	Chain-based (Polybot)
Rectangular - Homogeneous shapes					
Rectangular - Shapes are multiples of each other					
Rectangular - Arbitrary shapes					
Non-Rectangular					

Table 1: Framework for heterogeneous modules. We would like to fill in lower bounds analysis, algorithms and applications for each entry.

in the Sandia module, and the "deformable hexagon" [16] uses deformation. It is important to note that modules can implement motions other than their primary actuation mode using metamodules or other techniques.

Based on this framework we would like to develop abstract models of various module types for use in algorithm development and complexity analysis. As a starting point, we propose a theoretical module we call the *sliding cube* model. This idea is based on the 2D *Wang Tile* used by Rothemund and Winfree [55] to prove bounds for a related problem, self-assembling square units. A Wang Tile is a unit square with a *binding domain*, or glue, specified for each side chosen from a fixed alphabet Σ. The binding domain, along with a strength function, determine whether tiles bond with each other when placed together. Our problem requires three dimensions, and additional properties not included in the Wang Tile. These include actuation type and capabilities such as special sensors, power, or computation. To capture heterogeneity in our abstract model we use the following 3D extension of the Wang Tile:

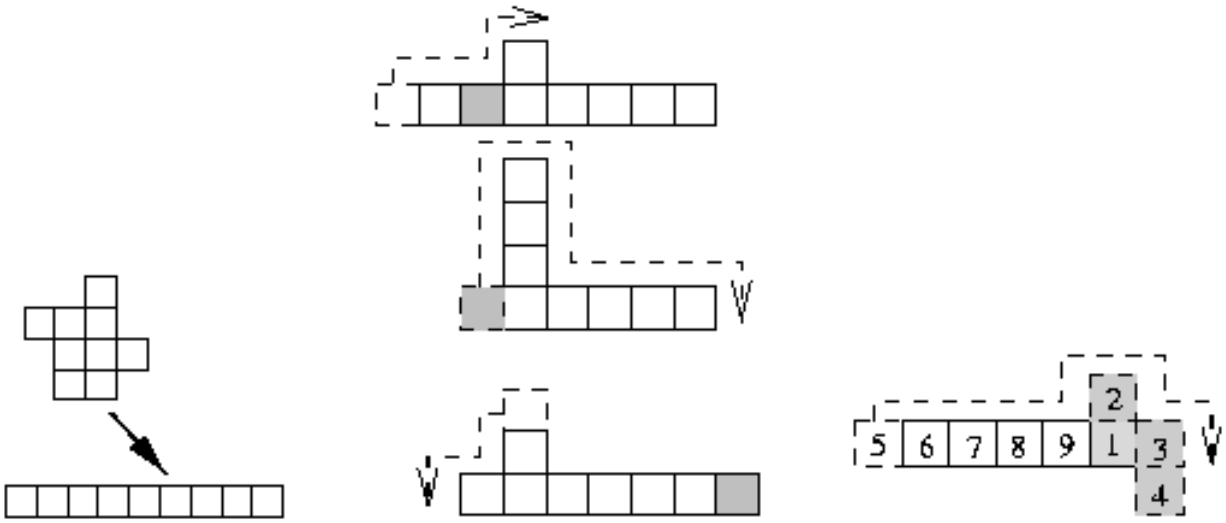
Sliding Cube Properties

- size (x,y,z): a rectangular prism
- type: unique class-level identifier chosen from alphabet of module types
- connectors: function maps each face to alphabet of connector types
- motion primitives: function maps current position to possible neighbor positions

The sliding cube is implementable by a number of modules in the matrix, such as rectangular Scaling and Rotational actuation systems. The downside is that metamodules are often required to implement the sliding actuation model. This is essentially the same motion used by recent Cellular Automata-style algorithms [8, 70], however, and at least one physical module directly implements this actuation mode [15]. We consider this to be a first attempt at this model, and intend to refine it as part of our proposed research.

4.2 Reconfiguration in the Sliding Cube Model

General heterogeneous reconfiguration planning is an open problem. Although heterogeneous reconfiguration planning seems similar to the Warehouse problem, recall that the Warehouse problem has a polynomial-time solution given sufficient free space. In reconfiguration planning, we have plenty of free space available,



(a) (b) (c)

Figure 4: Illustration of MeltSortGrow algorithm. The melt step is shown in (a), the shaded module is "unlocked" in (b), and (c) shows the final configuration being grown. The labels in (c) indicate the chosen assembly order, and shaded modules are their final position.

i.e. we do not assume the problem to be limited by any type of bounding rectangle, so we expect the heterogeneous reconfiguration problem to also be solvable in polynomial time. We outline a centralized version of such an algorithm in this section as an existence proof, and identify areas of improvement.

We will first attempt to adapt Sharma and Aloimonos's 2D algorithms to solve the reconfiguration problem in 2D and 3D. This implies a centralized algorithm using the sliding cube model described above. To begin, we will assume equal size modules and only assume unique identifiers differentiating modules. Sharma and Aloimonos present several variations of $O(n^2)$ time Warehouse algorithms with constrained free space distribution. The biggest difference between their tiles and the sliding cube is that primitive tile motion in their algorithm is not restricted to sliding along other modules as is ours. Another difference is their problem has no connectivity constraints.

The basic steps of the algorithm are sort, extract, assimilate, compact. The last three steps are iterated until the goal is reached. In the sort step, the tiles are basically melted into a line, similar to the MeltGrow reconfiguration algorithm. Then tiles are repeatedly removed from the sorted order, put in final position in the goal shape, and the sorted group is updated to remove the hole left by the removing the last tile. The challenge is to maintain connectivity while performing these operations.

A sketch of this algorithm, adapted to reconfiguration planning, is proposed in Algorithm 1. This is a centralized, planar reconfiguration algorithm for same-size Sliding Cubes with unique identifiers, meant to show a preliminary approach to the problem rather than to present an efficient solution. The idea is similar to the MeltGrow algorithm, although an additional sorting step is added to support heterogeneity. The first line of the algorithm, the "Melt" step, transforms the given configuration into a single line as in Figure 4, left. One approach is to repeatedly choose a module not already in position, and move it around the perimeter to the end of the line. The next step is to compute a sort order that allows the goal configuration to be assembled, or in other words, a label for each module that specifies the sequence in which it is assembled into the final reconfiguration. One difficulty is to maintain module connectivity during assembly; another

is to avoid "painting yourself into a corner" by choosing a sequence that intersects the current module chain. This step is difficult, but one way to generate the assembly sequence is to begin with the left-most module (choose any one if there are multiples) and then pick any adjacent unoccupied position, add it to the sequence, and consider that position filled. Repeat until the sequence is complete. Now the modules can be sorted using selection-sort. To select, or "unlock", a module in the middle of the chain, move all modules to its left into a stack on top of the module immediately to the right as illustrated in Figure 4, middle. This frees the selected module to move into position at the far right of the chain, and the stacked modules return to their original positions in the chain, shifted one to the right. At this point the chain is sorted left-to-right in assembly order. The final step of the algorithm is to build the goal configuration beginning at the right end of the chain as in Figure 4, right. The growing configuration does not intersect the chain since we started with the left-most module, and we can always position the next module while maintaining connectivity since our assembly order always picks an unoccupied position on the "crust" of the final configuration.

In analyzing the complexity of this algorithm, we assume that the output is a plan composed of a sequence of one-step module motions. Computation time is therefore at least as great as "execution time" in terms of the number of motions in the plan, so we equate running time with computation time. We further assume that the length of any module path is bounded by $O(n)$ steps. The melt and grow steps of the algorithm therefore run in $O(n^2)$ time, and the assembly order (as presented) can also be computed in quadratic time. The running time is dominated by sorting, however, which requires $O(n^2)$ steps for selection sort with $O(n)$ time required per step, or $O(n^3)$ overall.

Algorithm 1 Centralized 2D heterogeneous reconfiguration: MeltSortGrow.

```
1: "Melt" configuration into 1D chain
2: Compute feasible assembly order
3: Sort chain by assembly order
4: while Reconfiguration is not complete do
5:     Move next module into final position
```

We provide the MeltSortGrow algorithm as a generic skeleton that we intend to flesh out and prove correct for arbitrary reconfiguration. Although we may not be able to improve the asymptotic behavior, we would like to reduce the currently expansive free space requirements and large hidden constants. We would also like to extend the algorithm to 3D, and develop decentralized versions based on the same general approach.

No software exists to simulate the Sliding Cube, so we plan to construct a simulator in which to implement our algorithms. In the Goal Recognition problem, we built a Java3D simulation for homogeneous modules; this code will form the basis for the new simulator.

Finally, it is desirable to implement reconfiguration in hardware. The difficulty is that the Crystal modules are 2D, and due to connector problems are currently incapable of supporting the primitive reconfiguration operations. It might be possible to use a robot from a different lab, such as the Telecube, to perform experiments.

4.2.1 Increasing Heterogeneity

Sharma and Aloimonos's Warehouse algorithm variants show that essentially the same algorithm works with rectangular tiles of various size ratios, from equal to arbitrary. This leads us to propose to extend our solution to modules that are rectangular, but various sizes. Since the MeltGrowSort algorithm moves modules over the surface of the robot only, we believe that an extension to different module sizes is possible. Such an algorithm would allow self-assembly of rigid structures such as scaffolding. Simulations should use the engine built for the basic algorithm, with variable size modules.

The next step would be to relax the assumption that all modules have the same actuation mode. A simple yet useful variant would be to allow non-actuated modules. We explored path planning for such modules in our earlier work in self-repair, and we would like to extend these techniques and incorporate them into general reconfiguration algorithms.

4.3 Lower Bounds

A major question is how to determine lower bounds on the complexity of the heterogeneous reconfiguration problem. We wish to fill in our proposed matrix framework with complexity analyses. Our goal is to begin with the heavily constrained version of the reconfiguration problem described above, then relax assumptions to deal with greater heterogeneity. The model already handles assigning different capabilities to modules (through class ID's), so the next challenge would be to allow various shapes. Our initial approach is to relate the reconfiguration problem to the Sorting problem. The Warehouse problem algorithm essentially sorts modules by type, then assembles the new configuration. The Melt step of the MeltGrow algorithm can be though of as sorting as well, although there is only one module type.

4.4 Applications and Other Problems

We would also like to explore applications involving resource optimization problems in SR systems. Power is generally a major hurdle in developing physical reconfigurable systems. Most systems, in fact, are tethered. This presents a good opportunity to investigate how power can be distributed in the system to optimize performance. How could specialized, non-actuated battery modules be used in minimizing total current across small local connections? We plan to address this problem by investigating how to determine configurations where heavy modules (batteries) are kept at the base of the robot, and are distributed according to load. One approach is to use voltage drop across module connectors in a potential field algorithm.

A related application is the sensor deployment problem. Sensor modules should remain inside the body of the robot during locomotion, and should be deployed to the surface as necessary. Components to this problem include: determine when to hide or deploy the sensor, choose a configuration with the sensor in an appropriate position, and execute a suitable reconfiguration algorithm.

5 Completed Work in Reconfiguration Planning

We have previously investigated a number of topics in SR robotics that map into the reconfiguration planning category of our research agenda (see Section 3.2). For instance, *3D-MBP* is an algorithm for planning trajectories of modules in unit-compressible systems. This is used in the self-repair application, but also addresses reconfiguration with non-actuated modules and provides tools for handling resource trade-offs. Our distributed inchworm work presents reconfiguration applied to the locomotion application for unit-compressible systems, and our Goal Recognition work presents a distributed method for recognizing when a given configuration has been attained in a heterogeneous system. We describe the results in this section, and they can also be found in the literature [24, 23, 5, 6].

5.1 Rectilinear Shortest Paths Minimizing Turns: 3D-MBP

Much effort has been devoted to the important motion planning problem [42]. In motion planning for unit-compressible lattice-based self-reconfigurable robots, modules move through the structure of the robot so as to generate a desired shape change for the robot as a whole. Because modules in these robots form a lattice, module motion is rectilinear. Moreover, changing the direction of movement of a module is much more time consuming than maintaining straight-line motion, so efficient planning involves finding a rectilinear

path among obstacles with a minimum number of bends. Here we describe our solution to this problem, apply this solution to the self-repair problem for unit-compressible systems, and describe how the result is a heterogeneous reconfiguration planner with two module types.

The general problem of finding a shortest path between two points among obstacles has been extensively studied in many contexts, and with common variations such as the rectilinearity constraint and bend-distance[3] metric we consider here. Optimal solutions exist for finding a minimum-bend path (MBP) in 2D, but much less work has been done with the 3D variant, even though many applications require a 3D solution. For 2D, the optimal $O(e \log e)$ running time [19], where e is the number of obstacle edges, has been achieved by a number of algorithms [43]. However, these do not extend trivially to higher dimensions. In Fitch, Butler and Rus [23], we present an algorithm to solve the 3D minimum-bend-path problem (3D-MBP), and apply the algorithm to motion planning for self-repair in self-reconfigurable robots. The algorithm runs in $O(n^2 \log n + n^2 I)$ time, where n is the number of obstacle vertices and I is the maximum number of obstacles intersected by a line parallel to the x axis. This result, to our knowledge, is the first published 3D-MBP solution, and is the first to extend the wavefront technique to 3D.

The goal in 3D-MBP is to find a rectilinear path with fewest bends from a source point s to a destination d in $\Re^3$ among obstacles. A rectilinear path is a path composed of a series of connected line segments, each parallel to one of the coordinate axes. In this case, we consider only orthohedral obstacles, which have all edges parallel to one of the coordinate axes. The main idea of our solution is to explore the 3D environment using linear wavefronts that sweep in one of two orientations. Wavefronts spawn children according to certain rules, and eventually the entire problem space is searched.

Our algorithm extends a 2D algorithm by Lee [43] using the "continuous Dijkstra" approach. The basic approach is to maintain distance labels on obstacle vertices such that a label represents an upper bound on the length of a shortest path from the source to the labelled vertex. In each iteration we expand the labelled region by exploring a small area away from a vertex with a minimum distance label. In this extension of the horizontal wavefront technique, a wavefront is still a line segment but may travel in one of two possible directions (y or z) and spawns child wavefronts in both directions accordingly. When a minimum-distance wavefront reaches the destination, we recreate the path by following predecessor pointers back to the source.

Conceptually, the wavefronts explore the surfaces of cells in an exact 3D cell-decomposition of the environment, instead of explicitly exploring cell interiors. It is easy to see that all paths through a given series of cells are homotopic, and that paths through the faces can be constructed that are just as good (in number of bends) as any path through the cell volumes. To explore all cell faces, we first take the 3D problem and create a set of 2D problems parallel to the xy-plane at the z coordinates of all obstacle vertices. Another set of 2D problems are created that are aligned with the xz-plane at obstacle y coordinates. Wavefronts can then be thought of as "living" in one of these 2D problems. Unlike in 2D, the sweep operation will not only generate child wavefronts in its own 2D problem, but may also spawn children in intersecting problems.

The main computation is given in Algorithm 2. To begin, we insert four wavefronts at the source: an up-going xy, a down-going xy, an up-going xz and a down-going xz. From there we remove a wavefront from the priority queue of existing wavefronts based on minimum bend-value, drag it, and insert all newly generated wavefronts as described below. When a wavefront hits the destination, we continue until all remaining wavefronts have minimum bend-values at least as large as the best.

A simple example is shown in Figure 5. In (a), the initial four wavefronts are inserted. Wavefront w_1 is taken from the queue and dragged in (b), generating child wavefronts as described below. Part (c) shows wavefront w_2 being dragged, also generating new wavefronts. Next, other wavefronts with zero minimum bends would be dragged, but for sake of illustration we skip directly to w_3, shown in (d) and (e) in a 2D detail. In (e), the destination is reached. The algorithm continues until all wavefronts have at least two bends (the best so far), then the path is generated, shown in (f).

[3] Many authors use the term *link-distance*.

Algorithm 2 3D-MBP

1: Preprocess to generate set of 2D problems
2: Let Q be a priority queue sorted by minimum bend distance
3: Insert four waves into Q at s: up xy, down xy, up xz, down xz.
4: **while** Q is not empty and d is unmarked **do**
5: $w = DeleteMin(Q)$
6: Drag w from event point p to nearest event point p'
7: Apply 2D wavefront operations to w
8: Mark p'
9: Insert into Q counter-oriented child wavefronts w_i at intersection points between p and p', in both UP and DOWN directions
10: **if** d is unmarked **then**
11: Fail
12: **else**
13: let $b_{min} =$ distance(d)
14: **while** Q is not empty and min(Q) $< b_{min}$ **do**
15: Sweep as in lines 5-9
16: If wavefront reaches d, update b_{min}
17: Generate path P by following pointers from d to s
18: Output b_{min} and P

In Fitch, Butler and Rus [23], we show the following results:

Theorem 1. *The algorithm 3D-MBP correctly finds a minimum-bend path from s to d.*

Theorem 2. *The algorithm 3D-MBP runs in $O(n^2 \log n + n^2 I)$ time, where n is the number of obstacle vertices, and I is the maximum number of obstacles intersected by a line parallel to the x axis.*

Proofs of these theorems, along with further discussion of the data structures, preprocessing, and other details of the algorithm are also provided.

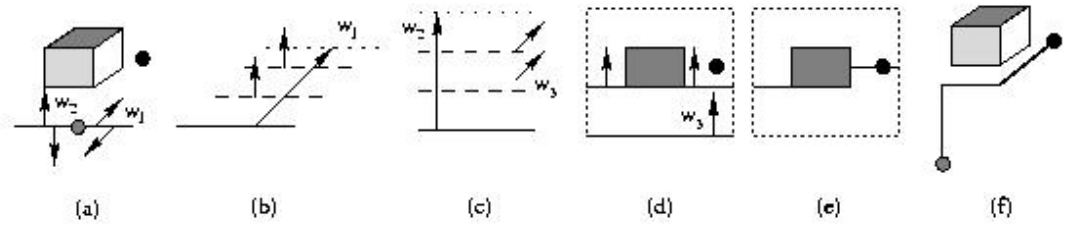


Figure 5: Computing a simple 3D path. In (a), four wavefronts are inserted (one in each possible direction). Part (b) shows w_1 being dragged, along with the resulting child wavefronts. Similarly, w_2 is dragged in (c). Parts (d) and (e) are 2D detail views of w_3 as it is dragged. The final path is shown in (f).

5.1.1 Applying 3D-MBP to Self-Repair

The self-repair problem is to restore functionality to a SR robot, without outside intervention, in response to module failure. Our strategy is to detect the failure, eject the failed module from the system, and replace

it with a spare built into the robot's structure. Path planning in this problem is based on finding rectilinear paths that minimize the number of turns. Our 3D-MBP algorithm addresses this issue, and in this section we describe how 3D-MBP is applied to the self-repair application. This technique is also a type of reconfiguration algorithm for systems with limited heterogeneity, and is described below.

Motion planning in the Crystal based on virtual module relocation reduces to finding a rectilinear path through the robot structure. Each segment of the path can be executed in constant time, assuming no failed modules, so an efficient motion plan requires a rectilinear path of minimum bends. Replacing a failed module (filling a "hole" in the structure) can be solved using virtual module relocation. To eject a failed module, this planning technique can not be used directly since here a particular module must be actually pushed (or pulled) to a position on the surface of the robot. However, pushing gaits to move the failed module, such as the one shown in Figure 6, also exhibit the property that turns are more expensive than straight line motion. Finding a minimum-bend path is therefore useful in both steps. An MBP problem is constructed by modelling the source and destination points in module coordinates, and holes and concavities in the structure as obstacles. Then, given a path, motion planning can be accomplished by iterating the appropriate gait over the path.

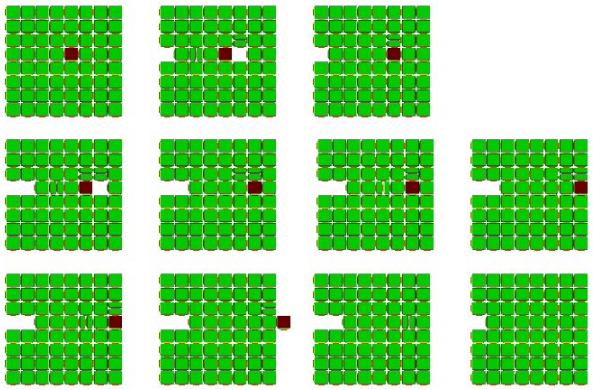


Figure 6: Pushing a failed module in a unit compressible system along a line. After initial setup, each push step requires four contractions/expansions. The second row illustrates two such steps. Finally, modules along the path are "reset" by shifting left.

This motion planning technique leads to a 2D self-repair solution [24], and easily extends to 3D given an efficient shortest path algorithm. A 3D rectilinear path can be decomposed into a sequence of 2D turns (not all of which are in the same plane). Therefore, given a 3D rectilinear path, a motion plan can be constructed by iterating the appropriate module gait over each path segment. Note that pushing gaits require a minimum amount of supporting structure, but we can build this into the path planning problem by growing the obstacles (holes in the structure and boundaries) by the required amount. This ensures that any path returned by the algorithm is feasible. We use the 3D-MBP algorithm to compute a path, and iterate a pushing gait along this path to eject the module. Finally, we use virtual module relocation to fill the gap in the structure left by the ejected module.

We have conducted experiments for 2D and 3D self-repair in Crystal robots in simulation. Output from our 3D-MBP implementation, in the form of a rectilinear path, is fed to a motion planning routine that generates motion primitives. These primitives are input in turn to our Crystal robot simulator that verifies physical feasibility and renders the simulation. Figure 6 outlines a pushing gait, and Figure 7 illustrates a sample simulation of a Crystal robot executing the pushing gait computed from a path returned by our 3D-MBP algorithm.

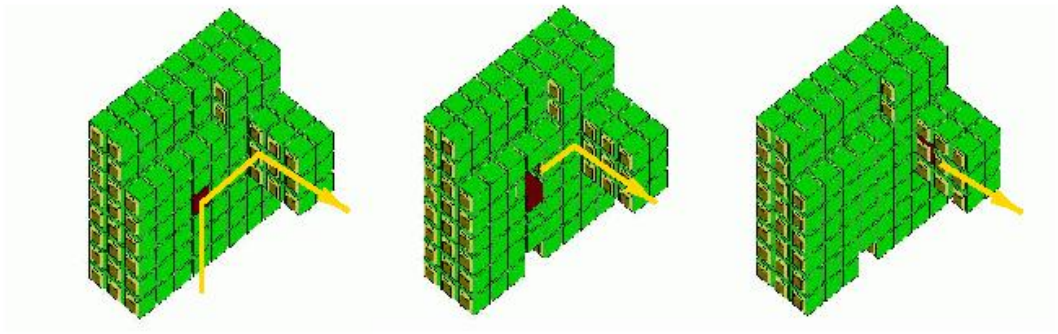


Figure 7: Cut-away view of 3D module ejection. The path is shown by the white arrow, and the failed module is darkened. The middle figure is previous to the first turn, and the last figure shows the failed module at the beginning of the final segment.

5.1.2 Reconfiguration with Limited Heterogeneity

The failed module in self-repair can be thought of as any non-actuated module, so the algorithms and cooperative gaits developed for the ejection step are one method of trajectory planning with non-actuated modules. Coupled with a higher-level planner for computing start and goal locations, the self-repair work enables a reconfiguration algorithm for systems with two classes of modules: actuated and non-actuated.

5.2 Reconfiguration for Locomotion

Reconfiguration is generally discussed in terms of task-specific shape transformation, but it can also be used for locomotion. We have developed a distributed locomotion algorithm for unit-compressible robots using inchworm-like motion, and implemented this algorithm in hardware on the Crystal system. We also performed extensive experimentation; the algorithm ran for over 75 hours in total at the SIGGRAPH and AAAI conferences. The algorithm and experiments are described in this section.

Inchworm locomotion uses friction with the ground to move a group of unit-compressible modules forward. The algorithm is based on a set of rules that test the module's relative geometry and generate expansions and contractions as well as messages that modules send to their neighbors. When a module receives a message from a neighbor indicating a change of state, it tests the neighborhood against all the rules, and if any rule applies, executes the commands associated with the rule. The algorithm is designed to mimic inchworm-like locomotion: compressions are created and propagated from the back of the group to the front, producing overall motion. See Figure 8.

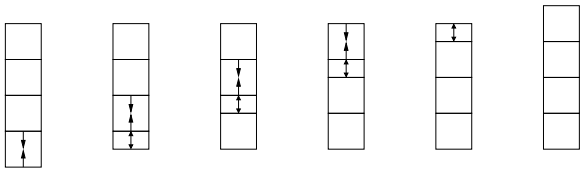
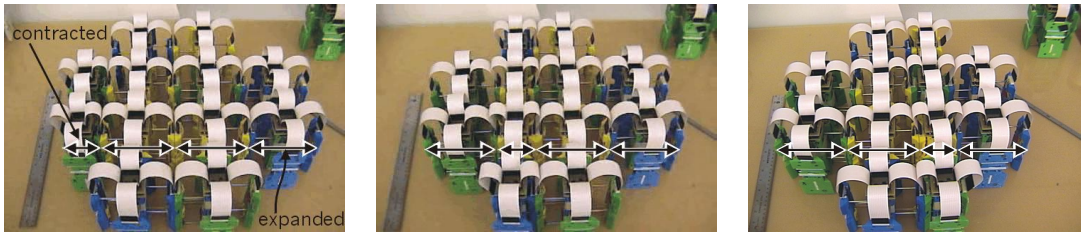


Figure 8: Schematic of module action under stand-alone locomotion, in which the group is heading upward, and the series (left to right) represents progress of a single inchworm "step".

The algorithm is presented as Algorithm 3. In the listing, we give a module's global state variables,

the message types it can send and receive, and the procedures that are called from the message handlers (including the rules of the algorithm). The "tail" module contracts first, which signals its forward neighbor to contract. Each module expands after contraction, so that the contraction propagates through the robot. When the contraction has reached the front of the group, the group will have moved half a unit forward (in theory; empirical results show nearly optimal distance-per-step for chains of five or more units [6]). Depending on context, once the leader of the group has contracted and expanded, it can then send a message back to the tail to initiate another step. We implemented this algorithm and performed experiments with various shapes, one of which is shown in Figure 9. The experiments successfully demonstrated reliable locomotion in the configurations we tested. See Butler, Fitch and Rus [6] for further discussion.

This locomotion gait is significant first in that it exemplifies the style of distributed, scalable algorithms we wish to develop and implement in proposed work. It also provides on possible gait to use in an application that chooses between various gaits, such as wheeled locomotion versus the inchworm, in response to the environment. In an extension to this algorithm used for demonstrations at the SIGGRAPH and AAAI conferences, we added touch sensors to the modules so that the head module could detect obstacles and reverse the direction of the inchworm. The result was that the robot "walked" back and forth between two obstacles on a table.



(a) (b) (c)

Figure 9: Photos of locomotion experiment for a blob shape. In (a), the leftmost column is contracted, and in (b) and (c) the following columns contract to make the group walk to the right.

5.3 Distributed Goal-Recognition

Since modular robots operate as a tightly coupled distributed system, most usage depends on the collective coordination and recognition of the current state of the system. This is especially important in the context of recognizing collectively that the system has achieved its goal. The general Goal Recognition problem asks whether a modular robot's configuration matches a particular goal shape. Systems originally considered are constructed from homogeneous modules and their configurations can thus be represented as a binary matrix, with 0 corresponding to empty space and 1 corresponding to space occupied by a module. Heterogeneous systems where modules are uniquely defined are supported as well by adapting the matrix data structure to use integers instead of bits. This representation leads to the following problem formulation: given an oriented goal matrix G and a modular robot A, determine if A's configuration matches that specified by G. We assume that the robot is a purely distributed system comprising modules that have local communication, limited processing power and memory, and that form a lattice. To simplify presentation, we consider square (in 2D) and cubic (in 3D) module shapes, but the algorithm works with other homogeneous shapes as well.

We developed a solution to distributed goal recognition in 2D and 3D based on a technique we call a *trace*. Intuitively, a trace is a tour of the modules of the robot matched at each step against the goal matrix. Our discussion is in terms of a homogeneous system, but the results also apply to a heterogeneous system with equal module sizes. Consider the situation where one module is assigned a position in the goal

Algorithm 3 Distributed stand-alone locomotion. Algorithm executes independently on each module.

State:
neighbors[], array of neighbors
heading, direction robot is moving: N,S,E,W

Messages:
inch (direction d), sent to move robot in direction d
  Action: set *heading* state to d, execute tryRules()
state (state s), announces state changes to neighbors
  Action: execute tryRules()

Procedures:
tryRules()
  position ← findPosition()
  if position = head **then**
    if neighbor[opposite(*heading*)] is contracted **then**
      contract, send state
      expand, send state
      send inch
  if position = body **then**
    if neighbors[opposite(*heading*)] is contracted **then**
      contract, send state
      expand, send state
  if position = tail and responding to *inch* message **then**
    contract, send state
    expand

findPosition()
  if rear neighbor but no forward neighbor **then**
    return head
  else if forward neighbor but no rear neighbor **then**
    return tail
  else
    return body

matrix. If the matrix has a 1 in that position, then the module is considered *valid*. That module then passes a message to a neighbor, including an indication of the matrix position corresponding to that neighbor. This new module then decides whether it is valid, and so on. If any modules are not valid, the trace is said to fail. Conversely, if all modules are valid, then the trace is said to succeed, and under certain conditions this implies that the robot is positively in the goal configuration. In particular, if the number of modules in the robot and in the goal matrix are the same, and both are fully connected, then a successful trace is both necessary and sufficient to solve the goal recognition problem.

The main idea of our algorithm, listed as Algorithm 4, is for multiple modules to initiate traces in parallel, each testing themselves against the same well-known position in the matrix (Figure 10). We call this position the *anchor*, and arbitrarily define it as the upper-left corner (in 3D, forward upper-left) of the target shape. The modules find this position by scanning the matrix for the first 1. In 2D, a module matching this local configuration would have no north or west neighbors, so any such physical module is called *special* and knows to initiate a trace when the algorithm begins. At most one trace will succeed, and the winning module then sends a global message. If all traces fail, however, then the situation is slightly more complex since no single module has enough information to discern global failure. If at least one module knows that all traces have failed, it can then propagate a global failure message. Therefore, when a trace fails, its originator sends a failure message that acts as a "meta" trace. Any special modules that receive this message hold it until their respective traces fail, then send as normal. When the module gets the meta-trace back, it knows that all other traces have failed and it can propagate the global failure message.

The correctness proof for this algorithm is detailed in [5]. In short, we show that traces will always return to the initiating module, and that comparing the actual robot perimeter to the goal always gives the correct result. We analyze the time and space requirements as follows. The total number of messages is $O(sk)$ trace messages(s traces of k messages each), where s is the number of special modules and k is the number of modules on the perimeter, plus $O(n)$ messages for propagating the solution, or $O(n + sk)$ total. Since $s < k <= n$, the overall upper bound on number of messages is $O(n^2)$. Since messages are sent in parallel, the time requirement is linear in the number of modules. The space requirement is constant per module, or linear overall. See [5] for a detailed explanation of the algorithm along with correctness proofs and analysis.

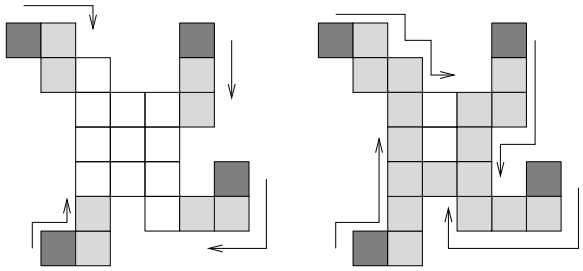
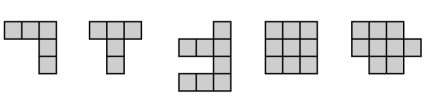


Figure 10: Multiple special modules. Dark modules are special and initiate traces in parallel, with paths of the traces indicated by arrows.

We implemented our algorithms in simulation, and implemented and executed the 2D Goal Recognition algorithm on the Crystal robot hardware using various configurations of the robot. Data is given in Table 2. Most trials were successful. Failures were due to failed connections in the initialization sequence. We hope to address this issue with a new connector design.

This result is useful for future algorithm development since it addresses a challenging issue for distributed algorithms: recognizing the achievement of a global property using local information. This is one component of a distributed solution to the reconfiguration problem. Since our Goal Recognition algorithm can work with heterogeneous systems, it will be valuable in developing solutions for heterogeneous reconfiguration.



Matches Goal	No	Yes	Yes	Yes	No
Trials	50	50	50	50	50
Successes	49	49	50	50	48
Message Count	25	16	67	32	44

Table 2: 2D Goal Recognition for various robot hardware configurations. Column headers represent tested robot shape. “Matches Goal” indicates whether the robot configuration matched the goal for that experiment. Message counts aggregate messages from all modules during simulations of the same module configurations.

Algorithm 4 2D Goal Recognition.

```
State:
special, (boolean) am I a special module
neighbors[], array of neighbors
goal, matrix representation of goal shape

Messages:
gr_start, begin goal recognition algorithm
   Action:
      PostMessage(gr_start)
      if special then
         LeftHandPass(trace, receivedFrom)
trace(vector goalPos), test my position against goal
   Action: HandleTrace(this message, goalPos)
trace_fail, trace from parent has failed at some point
   Action: HandleTraceFail(this message)
meta_trace, tests whether all traces have failed
   Action: HandleMetaTrace(this message)
gr_success, goal shape is matched
   Action: signal success
gr_failure, goal shape is not matched
   Action: signal failure

Procedures:
HandleTrace(Message m, vector goalPos)
   if m is from me then
      PostMessage(gr_success)
   else
      if valid(goalPos) then
         LeftHandPass(trace (neighborPosition, receivedFrom))
      else
         LeftHandPass(trace_fail, receivedFrom)

HandleTraceFail(Message m)
   if m is from me then
      LeftHandPass(meta_trace, receivedFrom)
   else
      LeftHandPass(trace_fail, receivedFrom)

HandleMetaTrace(Message m)
   if m is from me then
      PostMessage(gr_failure)
   else
      if special then
         while my trace has not returned do
            wait for my trace message
      LeftHandPass(meta_trace, receivedFrom)

PostMessage(Message m)
   send m to all neighbors

LeftHandPass(Message m, Sender s)
   send m to first clockwise neighbor after s
```

6 Conclusion

Like homogeneous systems, heterogeneous SR systems promise versatility and usefulness superior to fixed architecture robots through their ability to match structure to task. In addition, heterogeneous systems further this goal with their ability to match *capability* to task. The original vision of reconfigurable systems was inherently heterogeneous, and during the subsequent fifteen years researchers have accrued much knowledge of homogeneous systems. In this thesis, we propose to widen this understanding into the realm of heterogeneous systems. We plan to address fundamental algorithmic issues and demonstrate solutions in simulation and hardware where possible. The results of this work should shed light on the relative complexity of hardware versus software design in SR systems and lead to an algorithmic basis for heterogeneous self-reconfiguring robots.

We have proposed a framework for categorizing SR modules, and we have chosen a simple theoretical module on which to build reconfiguration algorithms. We will attempt to prove lower bounds for the basic problem and extend the results to systems with greater heterogeneity. There are other algorithmic issues we will address which are enabled by previous reconfiguration solutions, and by our previous work with

non-actuated modules, path planning, Goal Recognition, and distributed locomotion.

Finally, we propose to construct a software simulator with which to demonstrate our algorithms. This simulator should be suitable for further use by other researchers in the area. We also hope to perform hardware experiments where available.

6.1 Expected Contributions

The main expected contribution of this proposal is an algorithmic basis for heterogeneous SR systems. This contribution is supported by the following items:

- Framework for heterogeneous modules
- Reconfiguration in 2D and 3D with Sliding Cube model, with arbitrary size ratios
- Reconfiguration with non-actuated modules
- Complexity analysis for reconfiguration
- Applications involving resource trade-offs and optimization
- Implementation in simulation
- Hardware experimentation

6.2 Thesis Timeline

Fall 2002	Finalize theoretical model. Develop Basic Reconfiguration algorithm and lower bounds analysis. Begin simulator development
Winter 2002	Finish simulator and reconfiguration algorithm implementation. Extend algorithm to more heterogeneous systems. Perform hardware experiments
Spring 2003	Power/Communication application with simulation and experiments. Write thesis.

References

[1] L. Adleman. Towards a mathematical theory of self-assembly. Technical Report 00-722, University of Southern California, 2000.

[2] G. Beni. The concept of cellular robotic system. In *Proc. of IEEE International Symposium on Intelligent Control*, pages 57–62, 1988.

[3] G. Beni and J. Wang. Theoretical problems for the realization of distributed robotic systems. In *Proc. of IEEE International Symposium on Intelligent Control*, pages 1914–1920, 1991.

[4] Z. Butler, S. Byrnes, and D. Rus. Distributed motion planning for modular robots with unit-compressible modules. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, 2001.

[5] Z. Butler, R. Fitch, and D. Rus. Distributed goal recognition algorithms for modular robots. In *Proc. of IEEE ICRA*, 2002.

[6] Z. Butler, R. Fitch, and D. Rus. Experiments in distributed control for modular robots. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems, to appear*, 2002.

[7] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Cellular automata for decentralized control of self-reconfigurable robots. In *ICRA 2001 Workshop on Modular Self-Reconfigurable Robots*, 2001.

[8] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for a class of self-reconfigurable robots. In *Proc of IEEE ICRA*, 2002.

[9] Edited by T. Balch and L. Parker. *Robot Teams: From Diversity to Polymorphism*. A K Peters, Ltd., 2002.

[10] A. Cai, T. Fukuda, F. Arai, T. Ueyama, and A. Sakai. Hierarchical control architecture for cellular robotic system - simulations and experiments. In *Proc. of IEEE ICRA*, pages 1191–1196, 1995.

[11] Y. Cao, A. Fukunaga, and A. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4:1–23, 1997.

[12] A. Castano and P. Will. Mechanical design of a module for reconfigurable robots. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, pages 2203–2209, 2000.

[13] A. Castano and P. Will. A polymorphic robot team. In T. Balch and L. Parker, editors, *Robot Teams: From Diversity to Polymorphism*. A K Peters, Ltd., 2002.

[14] I. Chen and J. Burdick. Determining task optimal modular robot assembly configurations. In *Proc. of IEEE ICRA*, volume 1, pages 132 –137, 1995.

[15] C.-H. Chiang and G. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10(1):91–106, 2001.

[16] G. Chirikjian. Kinematics of a metamorphic robotic system. In *Proc. of IEEE ICRA*, pages 449–455, 1994.

[17] G. Chirikjian and A. Pamecha. Bounds for self-reconfiguration of metamorphic robots. In *Proc. of IEEE ICRA*, pages 1452–1457, 1996.

[18] S. Chitta and J. Ostrowski. Motion planning for hetrogeneous modular mobile systems. In *Proc. of IEEE ICRA*, pages 4077–4082, 2002.

[19] P.J. de rezende, D.T. Lee, and W.F. Wu. Rectilinear shortest paths in the presnece of rectangular barriers. *Discrete and Computational Geometry*, 4:41–53, 1989.

[20] S. Farritor and S. Dubowski. On modular design of field robotic systems. *Autonomous Robots*, 10(1):57–65, 2001.

[21] J. Fax and R. Murray. Graph laplacians and stabilization of vehicle formations. In *Proc. of IFAC World Congress*, 2002.

[22] J. Fax and R. Murray. Information flow and cooperative control of vehicle formations. In *Proc. of IFAC World Congress*, 2002.

[23] R. Fitch, Z. Butler, and D. Rus. 3D rectilinear motion planning with minimum bend paths. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, 2001.

[24] R. Fitch, D. Rus, and M. Vona. A basis for self-repair robots using self-reconfiguring crysta l modules. In *Intelligent Autonomous Systems 6*, 2000.

[25] T. Fukuda, M. Buss, and Y. Kawauchi. Communication system of cellular robot: Cebot. In *Proceedings of IECON '89: 1989 International Conference on Industrial Electronics, Control, and Instrumentation*, pages 695–700, 1989.

[26] T. Fukuda and T. Kaga. Distributed decision making of dynamically reconfigurable robotic system. In *Proc. of IEEE IROS*, pages 1604–1609, 1997.

[27] T. Fukuda and Y. Kawakuchi. Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator. In *Proc. of IEEE ICRA*, pages 662–7, 1990.

[28] T. Fukuda and S. Nakagawa. A dynamically reconfigurable robotic system (concept of a system and optimal configurations). In *Proceedings of IECON '87: 1987 International Conference on Industrial Electronics, Control, and Instrumentation*, 1987.

[29] T. Fukuda, S. Nakagawa, Y. Kawauchi, and M. Buss. Self organising robots based on cell structures - cebot. In *Proc. 1988 IEEE International Workshop on Intelligent Robots and Systems*, pages 145–150, 1988.

[30] T. Fukuda, T. Ueyama, and F. Arai. Control strategy for a network of cellular robots - determination of a master cell for cellular robotic network based on a potential energy. In *Proc. of IEEE ICRA*, pages 1616–1621, 1991.

[31] S. Hackwood and J. Wang. The engineering of cellular robotic systems. In *Proc. of IEEE International Symposium on Intelligent Control*, pages 70–75, 1988.

[32] J. Hopcroft, J. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; $pspace-$hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[33] K. Hosokawa, T. Tsujimori, T. Fujii, H. Kaetsu, H. Asama, Y. Koruda, and I. Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. of IEEE ICRA*, pages 2858–63, 1998.

[34] R. Hui, N. Kircanski, A. Goldenberg, C. Zhou, P. Kuzan, J. Wiercienski, D. Gershon, and P. Sinha. Design of the iris facility-a modular, reconfigurable and expandable robot test bed. In *Proc. of IEEE ICRA*, volume 3, pages 155 –160, 1993.

[35] N. Inou, M. Koseki, and H. Kobayashi. Pneumatic cellular robots forming a mechanical structure. In *TITech COE/Super Mechano-System Symposium 2001*, page 67, 2001.

[36] T. Kawauchi, M. Inaba, and T. Fukuda. A relation between resource amount and system performance of the cellular robotic system (cebot). In *Proc. of IEEE IROS*, pages 454–459, 1993.

[37] E. Klavins. Automatic synthesis of controllers for distributed assembly and formation forming. In *Proc. of IEEE ICRA*, pages 3296 –3302, 2002.

[38] K. Kotay and D. Rus. Locomotion versatility through self-reconfiguration. *Robotics and Autonomous Systems*, 26:217–32, 1999.

[39] K. Kotay and D. Rus. Scalable parallel algorithm for configuration planning for self-reconfiguring robots. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers*, Boston, 2000.

[40] K. Kotay, D. Rus, M. Vona, and C. McGray. The self-reconfiguring robotic molecule: design and control algorithms. In *Proc. of the Workshop on Algorithmic Foundations of Robotics*, 1998.

[41] J. Kubica, A. Casal, and T. Hogg. Complex behaviors from local rules in modular self-reconfigurable robots. In *Proc. of IEEE ICRA*, pages 360–7, 2001.

[42] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[43] D.T. Lee, C.D. Yang, and C.K. Wong. Rectilinear paths among rectilinear obstacles. *Discrete Applied Mathematics*, 70:185–215, 1996.

[44] W. H. Lee and A. Sanderson. Dynamic analysis and distributed control of the tetrabot modular reconfigurable robot system. *Autonomous Robots*, 10(1):67–82, 2001.

[45] S. Murata, H. Kurokawa, and S. Kokaji. Self-assembling machine. In *Proc. of IEEE ICRA*, pages 442–8, 1994.

[46] S. Murata, H. Kurokawa, E. Yoshida, K. Tomita, and S. Kokaji. A 3-D self-reconfigurable structure. In *Proc. of IEEE ICRA*, pages 432–9, May 1998.

[47] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. Hardware design of modular robotic system. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, pages 2210–7, 2000.

[48] R. Nagpal. *Programmable Self-Assembly: Constructing Global shape using Biologically-inspired Local Interactions and Origami Mathematics*. PhD thesis, Massachusetts Institute of Technology, 2001. AI Technical Report 2001-008.

[49] M. Nilsson. Why snake robots need torsion-free joints and how to build them. In *Proc. of IEEE ICRA*, pages 412–417, 1998.

[50] M. Nilsson. Heavy-duty connectors for self-reconfiguring robots. In *Proc. of IEEE ICRA*, pages 4071–4076, 2002.

[51] R. Olfati-Saber and R. Murray. Distributed cooperative control of multiple vehicle formations using structural potential functions. In *Proc. of IFAC World Congress*, 2002.

[52] A. Pamecha, C-J. Chiang, D. Stein, and G. Chirikjian. Design and implementation of metamorphic robots. In *Proc. of the 1996 ASME Design Engineering Technical Conf. and Computers in Engineering Conf.*, 1996.

[53] A. Pamecha, I. Ebert-Uphoff, and G. Chirikjian. Useful metrics for modular robot motion planning. *IEEE Trans. on Robotics and Automation*, 13(4):531–45, 1997.

[54] C. Paredis, H.B. Brown, and P. Khosla. A rapidly deployable manipulator system. In *Proc. of IEEE ICRA*, pages 1434–1439, 1996.

[55] P. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *STOC 2000*, 2000.

[56] D. Rus and M. Vona. Self-reconfiguration planning with unit compressible modules. In *Proc. of IEEE ICRA*, pages 2513–20, 1999.

[57] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with unit-compressible modules. *Autonomous Robots*, 10(1):107–24, 2001.

[58] K. Saitou and M. Jakiela. Subassembly generation via mechanical conformational switches. *Artificial Life*, 2:377–416, 1995.

[59] B. Salemi, W.-M. Shen, and P. Will. Hormone-controlled metamorphic robots. In *Proc. of IEEE ICRA*, 2001.

[60] G. Sandini. Cellular robotics - annotated bibliography. Technical Report TR 1/92, LIRA-Lab - DIST University of Genova, Via Opera Pia 13 - 16145 Genova, Italy, July 1992.

[61] R. Sharma and Y. Aloimonos. Coordinated motion planning: The warehousman's problem with constraints on free space. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(1):130–141, 1992.

[62] W.-M. Shen, B. Salemi, and P. Will. Hormones for self-reconfigurable robots. In *Proc. of IAS-6*, 2000.

[63] W.-M. Shen, P. Will, and A. Castano. Robot modularity for self-reconfiguration. In *SPIE Conf. on Sensor Fusion and Decentralized Control in Robotic Systems 2*, 1999.

[64] K. Støy, W.-M. Shen, and P. Will. Global locomotion from local interaction in self-reconfigurable robots. In *Proc. of IAS-7*, 2002.

[65] J. Suh, S. Homans, and M. Yim. *Telecubes*: Mechanical design of a module for self-reconfiguring robotics. In *Proc of IEEE ICRA*, 2002.

[66] K. Tanie and H. Maekawa. Self-reconfigurable cellular robotic system. US Patent 5361186, 1993.

[67] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji. Self-assembly and self-repair method for a distributed mechanical system. *IEEE Trans. on Robotics and Automation*, 15(6):1035–45, Dec. 1999.

[68] C. Ünsal and P. Khosla. I(ces)-cubes: a modular self-reconfigurable bipartite robotic system. In *Proc. of SPIE*, volume 3839: Sensor Fusion and Decentralized Control in Robotic Systems II, pages 258–269, 1999.

[69] Cem Ünsal and Pradeep Khosla. Mechatronic design of a modular self-reconfiguring robotic system. In *Proc. of IEEE ICRA*, pages 1742–7, 2000.

[70] S. Vassilvitskii, M. Yim, and J. Suh. A complete, local and parallel reconfiguration algorithm for cube style modular robots. In *Proc. of IEEE ICRA*, 2002.

[71] M. Vona. A two dimensional crystalline atomic unit modular self-reconfigurable robot. Undergraduate Honors Thesis, Dartmouth College, 1999.

[72] M. Yim. A reconfigurable modular robot with multiple modes of locomotion. In *Proc. of JSME Conf. on Advanced Mechatronics*, Tokyo, 1993.

[73] M. Yim. New locomotion gaits. In *Proc. of IEEE ICRA*, pages 2508–2514, 1994.

[74] M. Yim. http://www2.parc.com/spl/projects/modrobots/chain/polybot/demonstrations/g2_spider.jpg. WebSite, June 2002.

[75] M. Yim, J. Lamping, E. Mao, and J.G. Chase. Rhombic dodecahedron shape for self-assembling robots. SPL TechReport P9710777, Xerox Palo Alto Research Center, 1997.

[76] M. Yim, Y. Zhang, J. Lamping, and E. Mao. Distributed control for 3D shape metamorphosis. *Autonomous Robots*, 10(1):41–56, 2001.

[77] E. Yoshida, S. Kokaji, S. Murata, H. Kurokawa, and K. Tomita. Miniaturized self-reconfigurable system using shape memory alloy. In *Proc. of the Int'l Conf. on Intelligent Robots and Systems*, pages 1579–1585, 1999.

[78] E. Yoshida, S. Murata, A. Kaminura, K. Tomita, H. Kurokawa, and S. Kokaji. Motion planning of self-reconfigurable modular robot. In *Proc. of Int'l Symposium on Experimental Robotics*, 2000.

[79] E. Yoshida, S. Murata, K. Tomita, H. Kurokawa, and S. Kokaji. An experimental study on a self-repairing modular machine. *Robotics and Autonomous Systems*, 29:79–89, 1999.